Abril

2019

Arkadium

TRABAJO DE FIN DE GRADO

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

AUTOR: DOMINGO JESÚS DE LA MATA GARCÍA

DIRECTOR: JORDI FORNÉS DE JUAN

Contenido

1. Introducción y contexto

Durante la década de los 90 las ciudades de todo el mundo se llenaron de salones arcade. Lleno hasta los topes de docenas de máquinas de videojuegos de todo tipo y género que funcionaban utilizando monedas para conseguir partidas. Parte de la infancia de muchos de los adultos actuales se ha dado dentro de este tipo de salones. Por lo que es normal que surja un movimiento o fenómeno basado en la nostalgia de quienes vivieron aquella "época dorada" de las máquinas arcade.

A lo largo de los últimos 30 años la industria de los videojuegos ha crecido y cambiado en muchos aspectos. Las máquinas arcade prácticamente han desaparecido y con ellas los propios salones arcade, exceptuando Japón donde hay una cultura de los salones arcade muy arraigada en la sociedad japonesa, tanto en adultos como en jóvenes. Es el único país donde las empresas desarrolladoras de hardware y software dedican esfuerzos a este tipo de entretenimiento mientras que por lo general el PC y las consolas portátiles y de sobremesa dominan el mercado a nivel mundial.

Dado el dominio de PC y consolas en el terreno de los videojuegos, las pocas maquinas arcade que sobreviven lo hacen explorando otro tipo de jugabilidad con el uso de periféricos que no pueden encontrarse en otras plataformas como por ejemplo los simuladores de conducción de motos o algunos de los juegos de bailes más conocidos.

Podemos afirmar que las máquinas clásicas de 2 jugadores con joystick y botones desaparecieron del mercado hace mucho y aunque puede quedar en algún lugar algún dispositivo rezagado, están prácticamente extintas.

A lo largo de los años algunos de los juegos más populares de este tipo de máquinas arcade han recibido la categoría de clásicos atemporales como Street fighters 2, The king of fighters, Metal slug o Windjammers entre otros. Hasta el punto de reaparecer en las generaciones actuales, en ocasiones, más de 30 años después.

Esta categoría de títulos atemporales junto con el efecto de la nostalgia y una maravillosa comunidad de desarrolladores a nivel mundial ha dado lugar a diferentes softwares de emulación de todo tipo de máquinas arcade y consolas portátiles y de sobremesa para casi cualquier sistema operativo y arquitectura.

Además de todo lo anteriormente mencionado, desde hace más de 10 años se ha ido extendiendo una tendencia o filosofía en todo el mundo a través de internet denominada DIY

(Do It Yourself), Hazlo tú mismo. Esta tendencia ha sido impulsada enormemente por diferentes foros y comunidades de youtubers.

La unión de todos los conceptos anteriores ha dado lugar a que se formen comunidades de usuarios tratando de crear su propia maquina arcade y con ello también ha aparecido software para ello. FrontEnds, emuladores y demás herramientas para catalogar juegos o descargar las miniaturas correspondientes de cada uno de ellos todo con el fin de replicar el comportamiento de forma más o menos fiel de las máquinas de los años 90. En ocasiones incluso incluyen monedero, la ranura por la que se introducen las monedas.

A raíz de este software y herramientas nacen también diferentes sistemas operativos orientados a este tipo de dispositivos, aunque con el paso del tiempo se alejan cada vez más de la experiencia arcade en beneficio de una experiencia multimedia dado su uso en dispositivos populares ARM.

Arkadium tiene el objetivo de utilizar las tecnologías ya existentes como Linux, MAME y emulationStation para generar una nueva distribución capaz de simular el uso de una máquina arcade multijuego de forma fiel a la experiencia y uso de las máquinas arcade clásicas.

2. Estado del arte y antecedentes

2.1. Historia de las máquinas arcade

2.1.1. Antecedentes

En 1952, Alexander S. Douglas programó “OXO” como parte de una investigación en la universidad de Cambridge sobre la interacción de las máquinas con el hombre. “OXO” es considerado por algunos el primer juego con interfaz gráfica de la historia y por tanto el precursor de los videojuegos actuales y en particular de las máquinas arcade.

Posteriormente nacerían títulos como “Tennis for two” en 1958, creado por William Higginbotham y “Space war!” en 1962, de la mano de Steve Rushell, Martin Graetz y Wayne Wiitanen que sentarían las bases de lo que serían posteriormente los archiconocidos “PONG” y “asteroids”[1].

A principio de la década de los 70 se instaló en la universidad de Stanford el primer videojuego comercial del que se tiene constancia. Bill Pitts y Hugh Tuck desarrollaron “Galaxy Game”. La máquina se instaló en septiembre de 1971 con un coste de 20.000$. El precio por

partida era de 10 centavos y formaba colas de más de una hora para poder jugar. Fue retirada en 1979 y actualmente se encuentra en el museo de las computadoras de California. Esta es considerada la primera máquina arcade de la historia [2].

También en 1971, apareció “Computer space” que fue la primera máquina arcade comercial fabricada a gran escala. La máquina la diseñó Nolan Bushnell y la comercializó Nuttin Asociates. Aunque no fue un gran éxito comercial, sirvió para fundar Atari, que más adelante se convertiría en una de las más grandes del sector [3].

El primer desarrollo de Atari, y también su primer éxito comercial, fue “PONG”, en 1972. Cuentan que Nolan Bushnell dejó la primera máquina de “PONG” en una gasolinera y que cuando volvió a ver qué tal había ido, la máquina estaba tan llena que no podía funcionar [4].

Posteriormente aparecieron “Tank” en 1974 de la mano de Atari y Kee Games y “Gunfight” en 1975 desarrollado por Midway.

En 1976 aparecieron “Breakout”, juego que sentó las bases de lo que más tarde conoceríamos como “Arkanoid”, y “Night driver”, que fue el primer juego de conducción en primera persona de la historia. Ambas de la mano de Atari.

2.1.2. La edad de oro de las máquinas arcade

La conocida edad de oro comprende el lustro entre 1979 y 1984. Las mejoras técnicas de la época trajeron consigo mejoras en la capacidad gráfica de las máquinas y en el sonido. Estas mejoras permitieron el desarrollo de juegos que sentaron bases históricas y géneros que siguen vigentes hoy en día.

Durante este periodo la popularidad de las máquinas arcade creció exponencialmente hasta el punto en que era muy habitual encontrar este tipo de dispositivos de entretenimiento especialmente en bares y aparecieron los primeros salones arcade.

Por último, pero no menos importante, se comenzaron a asentar lo que serían los controles, más o menos standard de las cabinas. Una palanca y entre 4 y 6 botones. Aunque muchas máquinas han destacado por controles inmersivos o innovadores, cualquier persona que haya convivido con la edad de oro relaciona el característico joystick y sus botones con las máquinas arcade.

En 1978 apareció posiblemente el juego más conocido de la historia. De la mano de Taito nace “Space Invaders” inspirado por la guerra de los mundos, según Toshihiro Nishikado. Este se hizo tan popular que se produjo un problema grave en Japón relacionado con la escasez de monedas de 100 yenes. Al final incluso fue necesaria la intervención del gobierno que decidió acuñar más monedas [5].

Siguiendo el éxito de “Space invaders” en 1979 aparecieron “Asteroids” y “Galaxian”, dos referentes en la actualidad. Ambos, siguiendo la estela del juego de Taito consiguieron un hueco relevante en el mercado.

En 1980 nace el archiconocido “Pacman” a manos de Tohru Iwatani, diseñador de Namco. El nombre original del juego era “Puck-man” pero se convirtió en “Pacman” al exportarlo a estados unidos por posibles confusiones fonéticas.

También en 1980 aparece una máquina llamada “Deco Cassete” que, si bien no fue especialmente relevante en el mercado, sí lo es para este proyecto. La máquina proponía la posibilidad de cambiar de juegos mediante cintas.

Ya iniciada la década de los 80, allá por 1981 apareció el que sería probablemente el rey de las máquinas arcade durante los siguientes años. “Donkey Kong” fue el primer trabajo de Shigeru Miyamoto que, además permitió a Nintendo entrar en el mercado americano.

En 1982 nacería “Buck Rogers Planet of Zoom” que fue el primero en implementar un efecto 3D gracias a una nueva técnica.
Más tarde pero en el mismo año aparecería “Moon patrol” que tiene en su poder el título de primer videojuego de la historia en utilizar el efecto parallax.
El efecto parallax consiste en mover diferentes capas de la imagen a diferente velocidad para transmitir el efecto de perspectiva y profundidad de la visón humana.
Posteriormente y también en 1982 apareció “Pole position” que sentaría las bases de los juegos de carreras de la próxima década.

Ya en 1983 sale al mercado bajo el paraguas de Atari “I robot” el cual sería el primer juego comercial de la historia es mostrar gráficos vectoriales.
Los gráficos vectoriales son una técnica para representar objetos mediante formas geométricas primitivas.

También en el 83 aparece “Journey” que no fue ni popular ni exitoso, pero es recordado por ser el primer videojuego con gráficos digitalizados.

Posteriormente Cinematronics lanzó al mercado “Dragon’s Lair” en colaboración con un animador de Disney siendo de los primeros en utilizar el soporte DiscoVision. DiscoVisión era un formato de almacenamiento óptico.

Entre 1984 y 1985 aparecen las primeras máquinas con procesadores de 16 bits lo que supone un cambio considerable en los juegos de entonces y trajo además una considerable cantidad de títulos relevantes.

Así aparecieron en 1984 “Paperboy”, “Marble Madness” y “Pacland”. “Paperboy” destacaba en ser de los primeros juegos en disponer de un control peculiar, en este caso, el manillar de una bicicleta que hacía girar al avatar. “Marble Madness” utilizó dos trackball para el control de una bola por un laberinto en perspectiva isométrica, pero destacó por ser el primer juego en utilizar el “Atari system 1” y también el primero en utilizar sonido estéreo. “Pacland” fue relevante en este caso dado a su popularidad y su cambio al formato de plataformas.

Ya en 1985 aparecieron “Gauntlet”, “Gradius” y “Space Harrier”.
“Gauntlet” fue el primer juego que permitió 4 jugadores simultáneos en una misma partida. Esto supuso un éxito rotundo. Además de la temática similar a Dungeons and Dragons que estaba en pleno auge. “Gradius”, conocido en Europa como “Nemesis” introdujo la mecánica de recibir “upgrades” durante la partida en los juegos de scroll lateral de naves, mecánica que se popularizó increíblemente rápido. “Space Harrier” llegó al mercado bajo la tutela de Sega y fue el primer juego con cabina arcade con movimiento.

En 1986 llegaría “Out Run” realizado por Yu Suzuki para Sega. Era un juego de conducción que jugaba con la perspectiva y los mejores gráficos del momento. Además, fue uno de los primeros en permitir elección de fase en función de por donde cruzaban el checkpoint.

En 1987, “Shadowland” de Namco se coronó como el primer juego en utilizar gráficos de 16 bits. Pasó sin pena ni gloria.

“NARC” de Williams Electronics fue uno de los primeros en recibir ataques por parte de los padres por considerarse un juego ultraviolento en 1988. Hacía uso de gráficos digitalizados. Técnica que más adelante popularizaría “Mortal Kombat”.
También en 1988 apareció “Reikai Doushi” siendo el primero en utilizar en un videojuego la técnica calymation, que consiste en realizar las animaciones fotograma a fotograma con personajes de plastilina u otros materiales fácilmente maleables.

En 1988 también apareció Splatterhouse que fue el primer juego en tener Parental Advisory.

2.1.3. La decadencia de las máquinas arcade

Con la proliferación de las consolas de sobremesa, especialmente NES en 1985 y mega drive y super NES en 1988 y 1990 respectivamente, la popularidad de las máquinas y los salones arcade cayeron irremediablemente. Aún llegaron algunos juegos más que frenarían la caída y extenderían la vida de estos dispositivos durante algunos años más.
En la década de las 90 los principales responsables de evitar la prácticamente desaparición de las máquinas arcade fueron los juegos de lucha y Neo-Geo que siguió apostando por las arcade y lanzando títulos de considerable éxito al mercado.
De hecho, los juegos de lucha fueron tan populares que para la gran mayoría se han exportado los controles de las máquinas arcade a las consolas de sobremesa.

En 1991 llegaba “Street Fighter II”, el juego de lucha más icónico y popular, probablemente de la historia. Fue toda una revolución en muchos aspectos, gráficos, sonido, jugabilidad y competitividad y trajo colas de nuevo a los salones recreativos. Además de ser el primero de una larga ristra de juegos de lucha que seguirán su popularidad.

Posteriormente llegaron “Mortal Kombat” y “Mortal Kombat II” en 1992 y 1993 respectivamente. Juegos terriblemente polémicos en la época. Además, popularizó los modelos digitalizados que más adelante utilizaron otros títulos.

En 1993 apareció “Virtua Fighter” convirtiéndose en el primer juego de lucha en 3D del mercado. Seguido de “Killer Instinct” en 1994, el cual, se convirtió en el último juego importante de la lucha en arcade.

Posteriormente llegaron grandes sagas de Neo-geo como “Fatal Fury”, “Samurai Shodown”, “Metal Slug” o “The King Of Fighters” en una época en que el arcade perdía popularidad a marchas forzadas.

Si bien el mercado si ha ido reinventando sobre todo con periféricos particulares con éxito en algunos casos como “dance dance revolution” desde los años 2000 se puede decir que las máquinas arcades están en un momento muy bajo salvo en Japón donde la cultura local las mantiene en el mercado y donde tienen bastantes lanzamientos exclusivos de arcade [6].

2.2. La moda retro y el fenómeno DIY (Do It Yourself)

2.2.1. Redes sociales y comunicación

Desde la época dorada de las máquinas arcade el mundo ha cambiado enormemente. Especialmente en la forma de comunicarnos y relacionarnos debido a las mejoras de las diferentes tecnologías de la comunicación y el enorme auge de las redes sociales.

En 2004 apareció Facebook, aunque muy lejos de lo que es hoy en día. Creció rápidamente en popularidad y usuarios y sentó los precedentes de las nuevas formas de relación y comunicación que se desarrollarían.

Aunque internet ya era una herramienta donde algunas comunidades se unían para compartir gustos e intereses, las redes sociales hicieron muchísimo más accesible este tipo de fenómenos. El número de comunidades o grupos con afinidades comunes creció exponencialmente muchas comunidades pasaron de foros más o menos pequeños al mundo gigantesco que ofrecían las redes sociales. Esto permitió dar visibilidad a comunidades que ya existían.

En 2006 apareció Twitter que si bien a diferencia de Facebook tiene un paradigma de comunicación mucho más escueto y más centrado en compartir textos cortos de todo tipo a quien quisiera seguirte, demostró ser una herramienta increíblemente potente para dar visibilidad a tendencias de forma rapidísima y una plataforma muy eficiente para distribuir información entre un sector interesado [7].

En las últimas 2 décadas han ido apareciendo grupos y comunidades con cierto interés por los dispositivos y las modas retro. El impacto y la visibilidad de estos grupos y comunidades se ha magnificado de forma exponencial gracias a la ayuda de las redes sociales. Debido a esto, dichas comunidades han ido creciendo y la tendencia retro se ha ido expandiendo por todo el mundo [8].

La percepción de elementos genuinos propios de productos pertenecientes al pasado y especialmente la nostalgia parecen haber sido los principales desencadenantes del auge de la cultura de los años 70, 80 y 90.

Según el documento “Estudio de los factores de compra de productos retro y segmentación del mercado potencial retro” publicado por José S.Clemente Ricolfe, Juan M.Buitrago Vera y Eva Sendra Emper existen factores favorables para las empresas en el mercado retro y exponen como principales razones de consumo de este tipo de productos la

nostalgia, el reconocimiento o familiaridad, la autenticidad, percepción de una vía de escape hacia el pasado, refuerzo de la identidad personal, seguridad, calidad y la percepción de sentirse diferente o único [9].

2.2.2. Popularización del fenómeno DIY

No parece haber una fecha concreta en la que nace el fenómeno DIY, es un movimiento que fue ganando popularidad a lo largo de los años y se disparó con la ayuda de las redes sociales y la aparición de cientos de portales dedicados en exclusiva a ello.

Las siglas DIY son el acrónimo de “Do It Yourself” que, traducido a nuestro idioma significa “Hazlo tú mismo”. Se aplica a todo tipo de elementos desde simples posavasos hasta complejos mecanismos y productos.

En términos generales DIY promueve la acción de fabricar o ensamblar tus propios productos. Construir por ti mismo lo que necesitas, en vez de comprarlo ya completamente construido. Según el artículo “Understanding the do-it-yourself consumer: DIY motivations and outcomes”[10] las principales razones por las cuales un individuo decide emprender un Proyecto DIY son:

- Beneficio económico relativo (coste de producto muy alto respecto a los materiales).
- Ausencia de un producto de calidad.
- Ausencia de disponibilidad de un producto.
- Necesidad de personalización de un producto.

Llevar a cabo uno de estos proyectos consigue que las personas que los llevan a cabo se sientan capaces y construyen una identidad como “artesano”. Además, obtienen como resultado:

- Realización personal.
- Control sobre su obra.
- El placer de disfrutar de un proceso de artesanía.

Mientras que las primeras las primeras razones fomentan la entrada de nuevos miembros en las comunidades DIY, las segundas razones subjetivas fomentan que las personas que realizan algún proyecto DIY se mantengan en la comunidad y realizan más proyectos y muchos casos se convierten es “maestros” que documentan sus creaciones para que otros miembros más novatos los imiten.

Entre muchas otras cosas, una de las tendencias que fomentó el fenómeno DIY, especialmente junto con la aparición de RaspberryPI, es la de crear máquinas arcade caseras relativamente baratas.
Junto con esto, también aparecieron distribuciones que pretendían imitar el comportamiento de las máquinas arcade de las décadas de los 70, 80 y 90.

Alrededor de este fenómeno han aparecido grandes comunidades que tratan desde el corte de las piezas hasta la decoración pasando por controles, software y docenas de asuntos más.

2.3. Distribuciones y hardware para emular máquinas arcade

Con el paso de los años han surgido varias opciones populares tanto de software como de hardware. Algunas comerciales y otras de código libre. Alguna de ellas ha detenido su desarrollo mientras que otras siguen activas hoy en día y gozan de una comunidad muy amplia que mantiene vivos los proyectos. No vamos a ver todas las opciones, pero si las más importantes y populares a lo largo de los años y en la actualidad.

2.3.1. RetroOS

Nació como una modificación de WindowsXP. Salvando los posibles problemas legales que pueda tener, esta distribución fue muy popular entre 2008 y 2010. Utilizaba unas primeras versiones de HyperSpin. Más tarde, sobre 2012, apareció RetrOS 2.0.0 basado en Windows7 y con 2 frontEnd. HyperSpin y Maximum arcade. Ambos de código privativo y comercial.

Las versiones del sistema operativo fueron subidas de nuevo en 2015 en una nueva web donde el autor ha desarrollado un blog/negocio alrededor de las máquinas arcade. Aunque el sistema sigue disponible para continuar, su desarrollo está detenido.

Durante bastantes años fue la distribución más popular entre la comunidad. Quizás por la familiaridad que supone Windows para los usuarios respecto a otro tipo de sistemas.

2.3.2. Lakka

La primera versión de Lakka apareció en diciembre de 2010 pero la versión 1.0.0 descargable para el usuario llegó el 20 de octubre de 2011. Es un sistema operativo que nace

claramente orientado a la emulación de todo tipo de plataforma y no solamente de arcade. Pretende ser un sistema operativo para **consolas**.

El sistema operativo es open source y está basado en Linux. Utiliza libRetro como una capa de control que gestiona todos los emuladores disponibles con una interfaz común. Gestiona además los dispositivos gráficos, el input y el audio. De forma que permite hacer una abstracción casi completa del sistema. LibRetro es un conjunto de librerías open source que también se utilizan en otros sistemas que veremos más adelante y algunos otros desconocidos o menos relevantes.

Está especialmente orientado a dispositivos como RaspberryPi, ODroid y otros. De hecho tiene varias versiones diferentes pre compiladas en la web disponibles para descargar.

Utiliza retroArch como frontEnd. Éste, simula visualmente el menú de PlayStation 3 lo cual hace que sea muy cómodo trabajar especialmente si estás familiarizado con el entorno, pero aún si no lo estás resulta muy intuitivo aunque en su contra, no permite prácticamente personalización.

2.3.3. RecalBox

Aparentemente, la primera versión pública de recalbox es la versión 3.2.11 el 24 de marzo de 2015. Es un proyecto desarrollado inicialmente por el usuario de github/gitlab “digitalLumberjack” al que pronto se fueron uniendo diferentes desarrolladores. Actualmente, el desarrollador original ya no continúa implicado en el proyecto y ahora es desarrollado únicamente por la comunidad.

Recalbox nace como sistema operativo multiplataforma orientado a consolas, especialmente para RaspberryPi. De hecho, tienen una sección en su web denominada DIY Recallbox en las que venden RaspberryPi y controles compatibles.

Igual que Lakka, Recalbox utiliza libRetro para gestionar gráficos, input y sonido. Esta vez nos encontramos con un fork de EmulationStation como frontEnd. Un poco menos intuitivo que retroArch pero altamente personalizable. Hasta el punto de poder crear tus propios “temas” utilizando XML. Esto permite que los desarrolladores y colaboradores creen temas diferentes incluso para cada consola a emular lo que da un atractivo importante al sistema operativo.

Recalbox ofrece en sus versiones actuales varios reproductores multimedia, además de una versión de Kodi, una especie de frontEnd multimedia. Ofrece además varias herramientas de personalización de la experiencia incluyendo docenas de filtros que simulan el comportamiento de viejas pantallas.

2.3.4. Batocera.Linux

Batocera es literalmente un fork de RecalBox. El autor original de RecalBox, por alguna razón decidió abandonar el desarrollo, quizás no estaba de acuerdo con la dirección en que la comunidad llevaba el proyecto, y creó un fork denominado Batocera.Linux.

En términos estéticos y funcionales son tremendamente similares. Aunque lleva desde 2015 desarrollándose independientemente de RecalBox, de cara al usuario las diferencias son pequeñas.

Existen algunas pequeñas diferencias respecto a soporte para algunos periféricos y, según algunos comentarios que podemos encontrar en algunos foros sobre DIY arcade, Batocera consigue un mejor rendimiento en máquinas con pocos recursos.

En el resto de aspectos están dirigidos al mismo público y dan solución a los mismos problemas. RecalBox ha ganado bastante más popularidad en los últimos años.

2.3.5. Pandora Box

A diferencia del resto de opciones esta vez hablaremos, no solo de un software sino, de un producto completo que incluye hardware y software.

Podemos encontrar pandora box en 2 versiones. Una de ellas es un pequeño dispositivo similar a lo que podría ser una pequeña consola con conectores USB y HDMI para conectar la pantalla y los mandos y usarlo de hecho como una consola retro. La segunda versión es una tabla con controles arcade para dos jugadores, de un tamaño considerable, para conectar directamente a una pantalla. Esta segunda versión es mucho más popular que la primera.

No hay un único comerciante que explote el producto, sino un conjunto de fabricantes y distribuidores que venden este producto. Hay algunas pequeñas diferencias entre unos

fabricantes y otros, pero nada especialmente destacable más allá de pequeños cambios visuales.

Pandora Box incluye un sistema operativo preconfigurado con una interfaz muy sencilla que te muestra una lista de juegos disponibles, listo para pulsar y jugar directamente.

Una de las cosas más curiosas de PandoraBox es que incluye también los propios juegos. Lo cual, podría tener algunas consecuencias legales.

3. Alcance, justificación y utilidad

Los sistemas actuales más populares, especialmente Lakka, RecalBox y Batocera.Linux se desplazan cada vez más hacia el terreno multimedia y las funciones extra. Añadiendo constantemente nuevos emuladores de más y más plataformas y orientándose cada vez más a pequeñas consolas retro. Con conocimientos más o menos técnicos podrían personalizarse bastante, pero requiere un trabajo adicional.

Son sistemas operativos que además crecen en tamaño aun estando destinados, en muchos casos, a “pequeñas” tarjetas de memoria y cada vez van pesando más. Esto no se ha convertido en un problema debido al crecimiento en cuanto a almacenamiento de las tarjetas actuales.

RetroOS y PandoraBox sí llevan a cabo una simulación aproximada de la experiencia arcade, pero ambos son comerciales y privativos. De forma que no encontramos nada de software libre que pretenda simular una máquina arcade.

La comunidad que realiza proyectos DIY y crea este tipo de máquinas, en muchos casos no son expertos en software ni tienen conocimientos técnicos avanzados y se encuentran con tediosos problemas

3.1. Qué es Arkadium y qué pretende

Arkadium pretende, utilizando únicamente software libre, recrear el comportamiento de una máquina arcade. Fácil de usar para personas con mínimos conocimientos técnicos y lo más ligero posible. Compatible con la mayoría del hardware común en cuanto a controles

arcade. Inicialmente únicamente destinado a viejos equipos reciclados con arquitectura x86_64.

Los usuarios necesitan un sistema libre que les permita realizar una emulación específica de forma simple.

Para ello utilizaremos un conjunto de programas open source. Utilizaremos un kernel de Debian con un gestor de paquetes para facilitar las dependencias y versiones. Uniremos a los paquetes básicos de Debian, el programa de emulación de máquinas arcade MAME y el frontEnd EmulationStation.

MAME son las siglas de Multiple Arcade Machine Emulator, es un software multiplataforma para cientos de máquinas arcade diferentes. La primera versión disponible es de diciembre de 2007, por lo que nos encontramos con un software con más de 10 años de recorrido, cientos de forks diferentes. MAME además absorbió a MESS (Multi Emulator Super System) que amplió su número de plataformas a emular. Aunque al compilarlo puedes compilar MAME o MESS por separado sin problemas.

MAME dispone de un menú visual relativamente tosco en su uso, pero con muchas opciones y configuraciones permitiendo seleccionar la máquina a emular pero también se puede usar directamente como comando. Es open source y legal pero las ROMS, ficheros con el software a emular, generalmente están sujetas a derechos de autor. Existen ROMS libres de derechos y se pueden obtener los derechos de uso de algunos softwares.

EmulationStation es un software frontEnd, una capa visual que interactúa con el usuario y ejerce como lanzador de otros programas. En este caso, ofrece un menú muy personalizable con un aspecto visual, también muy personalizable que puede configurarse para que lance MAME con el ROM seleccionado.
Es también el frontEnd que utilizan Recalbox y Batocera.Linux. Debido a su simplicidad y su capacidad de personalización.

Utiliza ficheros xml para personalizar visualmente la interfaz, de forma que puede desarrollarse una personalización abstrayéndonos del funcionamiento del software. Esto a

fomentado que la comunidad cree docenas y docenas de temas y nos permitirá crear un aspecto visual único y propio para *Arkadium*.

3.2. Planificación y metodología

El proyecto *Arkadium* se desarrollará en su primera versión, únicamente para viejos equipos x86_64, en aproximadamente 4 meses. Durante el desarrollo de este hay muchos procesos que son dependientes de otros y se contará únicamente con una persona para llevar a cabo las tareas por lo que el trabajo con el software se hará de forma secuencia utilizando el método de cascada.

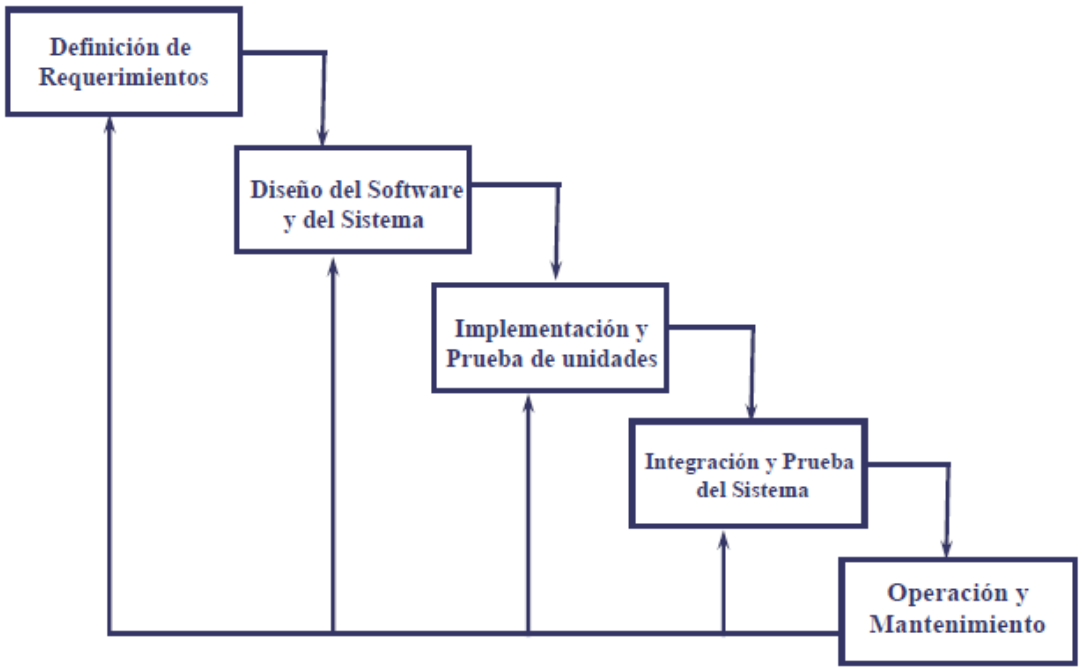

Figura 1: Metodología en cascada

Haciendo énfasis especialmente en el bucle Implementación e integración del software (Figura 1) dado que iremos integrando los diferentes softwares que formarán *Arkadium* de forma secuencial por fases.

A grandes rasgos el proceso será el siguiente. Instalación de una versión básica de Debian, descarga, compilación y configuración de MAME, pruebas del sistema con MAME, descarga, compilación, configuración, pruebas del sistema con EmulationStation,

personalización de EmulationStation y personalizar el sistema operativo. Por último, testing del sistema completo.

3.3. Posibles obstáculos y alternativas

A lo largo del desarrollo del proyecto pueden aparecer diferentes obstáculos e impedimentos tanto hardware como software. Y debemos tener diferentes alternativas o planes de acción para hacer frente en caso de aparecer.

3.3.1. Hardware

- Obstáculo: Fallo físico de alguno de los componentes.
  Alternativa 1: Prescindir del componente si no es imprescindible. Por ejemplo, un módulo de memoria RAM suponiendo que hay 2 disponibles.
  Alternativa 2: Adquisición de un nuevo componente compatible.

- Obstáculo: Incompatibilidad de alguno de los componentes con el software.
  Alternativa 1: Buscar soluciones vía software.
  Alternativa 2: Prescindir del componente si no es imprescindible.
  Alternativa 3: Sustituir el componente.

- Obstáculo: Inexistencia de drivers adecuados para los periféricos de los controles.
  Alternativa 1: Si hay documentación suficiente sobre el chip, programación del controlador.
  Alternativa 2: Sustitución del controlador.

3.3.2. Software

- Obstáculo: Problemas de compatibilidad con las versiones de las librerías.
  Alternativa 1: Actualizar los repositorios oficiales de Debian.

Alternativa 2: Utilizar otros repositorios.

Alternativa 3: Compilar directamente la versión adecuada.

- Obstáculo: Problemas con el kernel de Debian.

  Alternativa 1: Cambiar la versión del kernel.

  Alternativa 2: Utilizar el kernel de otra distribución.

  Alternativa 3: Compilar un kernel de Linux.

4. Planificación general del proyecto

El proyecto *Arkadium* tiene una duración de aproximadamente 4 meses iniciando las primeras pruebas e investigaciones el 10 de septiembre y finalizando el 10 de enero. Se aplicará una dedicación de 4 horas diarias, lo que supone 28 horas semanales, 420 horas en total.

La lectura del proyecto se realizará entre el lunes 28 de enero y el viernes 1 de febrero de 2019. Hay que entregar la memoria una semana antes según las normas del TFG por tanto, la entrega se debe realizar antes de 22 de enero. Esto permite un margen de aproximadamente semana y media para imprevistos y/o modificaciones.

Podría tenerse en consideración la opción de desplazar la solicitud del turno de lectura al turno de abril si los resultados, a mediados de diciembre (límite de petición de turno de lectura) los resultados están muy lejos de lo esperado. Como por ejemplo que no tuviese una versión funcional de *Arkadium*.

El proyecto lo llevará a cabo una sola persona y se llevará a cabo una realización secuencial de las tareas, empezando por el sistema operativo y finalizando por las aplicaciones clave de la distribución. Todas las tareas son dependientes de las tareas anteriores por lo que es buen proyecto para una sola persona y se desarrolla de forma natural utilizando metodología de cascada.

5. Recursos

A lo largo del desarrollo del proyecto utilizaremos diversos recursos de los que será imprescindible disponer para el correcto desarrollo.

5.1. Recursos humanos

Un único trabajador asumirá toda la carga y todos los roles implicados en el desarrollo del proyecto. Con una dedicación aproximada de 28 horas semanales.
En función del desarrollo de la vida personal y profesional del único partícipe del proyecto, es posible que aparezca una desviación respecto al diagrama de Gantt (Figura 2) pero se hará todo lo posible para ajustar el desarrollo del proyecto a la planificación.

Los 2 roles que se llevarán a cabo serán el de director de proyecto e ingeniero de sistemas. Aproximadamente 125 horas se realizarán asumiendo el rol de director de proyecto y las 295 horas restantes se llevarán a cabo como ingeniero de sistemas.

5.2. Recursos materiales

1) *Máquina arcade*. Construida por mí mismo, será el dispositivo en el que se implemente *Arkadium*.

- Procesador: Pentium D (x86_64)
- RAM: 1GB DDR2
- Monitor BenQ 19'' (4:3)
- Controles arcade de fabricación china.

2) *Disco duro externo (Maxtore) de 1TB*: Disco donde se harán todas las particiones necesarias para implementar *Arkadium*, también la partición de swap, Grub y el sistema operativo *Ubuntu*.

3) *Ubuntu*: Sistema operativo que se utilizará para compilar *Arkadium* hasta que sea bootable.

4) *Google drive*: Herramienta de almacenamiento en la nube para guardar copias de seguridad de toda la documentación, entregas de GEP y memoria, código fuente de las versiones de las librerías y cualquier otro documento con anotaciones.

5) *Correo electrónico Gmail*: Herramienta de comunicación que se usará durante la realización del proyecto para mantener el contacto con el director de este.

6) *Trello*: Herramienta software de control y planificación de tareas. Es una herramienta web que puede utilizarse desde cualquier equipo e incluso dispositivos móviles. El único requisito es conexión a internet.

7) *WPS office y Microsoft office*: Conjunto de herramientas de oficina que utilizaremos para escribir la memoria, el diagrama de Gantt y toda la documentación necesaria.

8) *Conexión a internet*: Necesaria para el uso de herramientas como Trello, Drive y los repositorios de Debian.

9) Conexión a la red eléctrica: Recurso totalmente necesario para el funcionamiento de todos los equipos necesarios para el desarrollo del proyecto.

6. Planificación del proyecto

6.1. Estimación de horas

Vamos a dividir el desarrollo del proyecto en etapas y asignar un número de horas para mantener un correcto seguimiento del desarrollo.

Tareas	Horas
Inicio del proyecto	**60**
Definición e investigación	40
Preparación del hardware necesario	20
Gestión del proyecto	**65**
Definición del alcance	15
Planificación temporal	10
Gestión económica	10
Reuniones y presentación GEP	15
Presentación final y entrega de la memoria	15
Preparación del entorno	**30**
Preparación de particiones	3
Instalación de Ubuntu	10
Preparación de particiones y fstab	7
Descarga de todos los paquetes necesarios	10
Compilación y creación sistema ficheros temporal	**70**
Compilación de programas	50

Instalación de programas	20
Instalación del sistema	**90**
Creación de los sistemas de ficheros virtuales	5
Compilación e instalación de dpkg	5
Descarga e instalación de paquetes .deb	30
Crear ficheros de configuración del sistema	10
Instalación y creación de usuarios	5
Descarga e instalación del kernel	10
Instalación y configuración de Grub	5
Instalación y configuración de librerías gráficas	20
Implementación de programas específicos	**80**
Compilar e instalar MAME	5
Configurar MAME	5
Compilar e instalar Emulation Station	5
Configurar Emulation Station	10
Personalizar Emulation Station	40
Configurar controles	15
Personalización de la distribución	**25**
Configurar autoarranque de Emulation Station	10
Configurar splash screen Arkadium	10
Configuración final de Grub	5
Total	**420**

Tabla 1: Planificación temporal

7. Diagrama de Gantt

	Arkadium dividido en semanas	
	Tareas	Horas
T1	**Inicio del proyecto**	**60**
T1.1	Definición e investigación	**40**
T1.2	Preparación del hardware necesario	**20**
T2	**Gestión del proyecto**	**110**
T2.1	Definición del alcance	**15**
T2.2	Planificación temporal	**10**
T2.3	Gestión económica	**10**
T2.4	Reuniones de control	**15**
T2.5	Documentación del proyecto	**60**
T3	**Preparación del entorno**	**30**
T3.1	Preparación de particiones	**3**
T3.2	Instalación de Ubuntu	**10**
T3.3	Preparación de particiones y fstab	**7**
T3.4	Descarga de todos los paquetes necesarios	**10**
T4	**Sistema de ficheros temporal**	**50**
T4.1	Compilación de programas	**30**
T4.2	Instalación de programas	**20**
T5	**Instalación del sistema**	**65**
T5.1	Creación de los sistemas de ficheros virtuales	**5**
T5.2	Compilación e instalación de dpkg	**5**
T5.3	Descarga e instalación de paquetes .deb	**10**
T5.4	Crear ficheros de configuración del sistema	**10**
T5.5	Instalación y creación de usuarios	**5**
T5.6	Descarga e instalación del kernel	**10**
T5.7	Instalación y configuración de grub	**5**
T5.8	Instalación y configuración de librerías gráficas	**15**
T6	**Implementación de programas específicos**	**80**
T6.1	Compilar e instalar MaMe	**5**
T6.2	Configurar MaMe	**5**
T6.3	Compilar e instalar Emulation Station	**5**
T6.4	Configurar Emulation Station	**10**
T6.5	Personalizar Emulation Station	**40**
T6.6	Configurar controles	**15**
T7	**Personalización de la distribución**	**25**
T7.1	Configurar autoarranque de Emulation Station	**10**
T7.2	Configurar splash screen Arkadium	**10**
T7.3	Configuración final de grub	**5**
	Total	**420**
	Imprevistos y alternativas	**92**

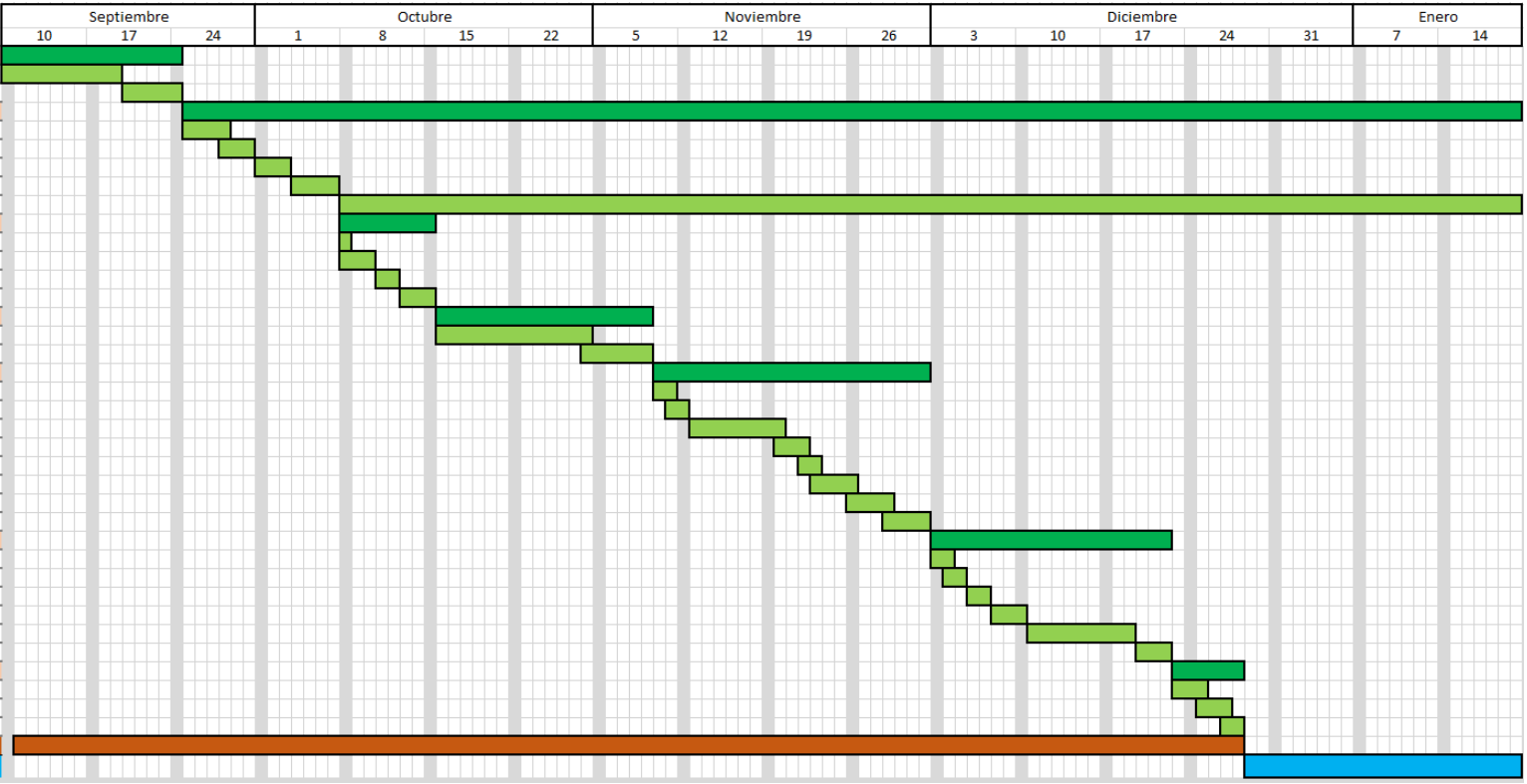

Figura 2: Diagrama de Gantt

8. Definición de las tareas

8.1. Inicio del proyecto

Definir el objetivo del proyecto y las tareas a desarrollar, recabar toda la información necesaria para formar una imagen clara del desarrollo de este. Definir la metodología de trabajo y los posibles inconvenientes y cómo lidiar con ellos. También detectar las posibles complicaciones externas, sociales o económicas que puedan aparecer.

8.2. Gestión del proyecto

Especificar y documentar la planificación y los procesos que se llevarán a cabo. Definir todo lo necesario desde los pequeños detalles hasta la organización de los puntos más críticos del proyecto.

8.3. Preparación del entorno

Definir el número y tipo de particiones necesarias para llevar a cabo el trabajo. Particionar el disco con lo especificado y dar el formato necesario a cada una de las particiones.

Instalar *Ubuntu* como sistema sobre el que se va a compilar el nuevo sistema. Montar los sistemas de ficheros de las nuevas particiones y especificarlas en fstab para hacerlo permanente de cara a trabajar a largo plazo.

Descargar los códigos fuente de los programas que necesitamos para hacer un sistema temporal y que compilaremos a posterior.

8.4. Compilación y creación sistema ficheros temporal

Compilación e instalación de los paquetes descargados posteriormente en un orden específico para evitar problemas de dependencias y dado que algunos programas se compilan por partes.

Binutils (1)	GCC (1)	Linux-4.18.5 API Headers	Glibc	Libstdc++	Binutils (2)
GCC (2)	Tcl	Expect	DejaGNU	M4	Ncurses
Bash	Bison	Bzip2	Coreutils	Diffutils	File
Findutils	Gawk	Gettext	Grep	Gzip	Make
Patch	Perl	Sed	Tar	Textinfo	Util-linux
Xz					

8.5. Instalación del sistema

Creación de los directorios necesarios en un sistema debían. Compilación del gestor de paquetes dpkg. Descargar los paquetes .deb que son dependencias de apt y apt. Crear los ficheros de configuración de red y de sistema. Instalación y creación de los usuarios necesarios. Descarga e instalación del kernel debían desde apt. Instalación y configuración de Grub y por último instalación y configuración de las librerías gráficas acorde al hardware.

8.6. Implementación de programas específicos

Descargar, compilar e instalar el emulador de juego arcade MAME. Configurar los directorios y las Bios de MAME.
Descargar, compilar e instalar el frontEnd EmulationStation. Configurar directorios, controles y aspecto visual de Emulation Station.

8.7. Personalizar la distribución

Configurar EmulationStation para que arranque automáticamente con el sistema. Configurar la pantalla de carga inicial del sistema para que aparezca el logo de *Arkadium*. Configuración de Grub para que no aparezca y arranque directamente *Arkadium*.

8.8. Dependencias de las tareas

TareaA > TareaB indica que TareaB depende de la realización de la TareaA.

- T1 > T2
- T2 > T3
- T3 > T4
- T4 > T5
- T5 > T6
- T6.1 > T6.4
- T6 > T7.1
- T5 > T7.2

9. Valoración de alternativas

Por la naturaleza del proyecto, el carácter crítico de funcionamiento del sistema operativo (Debian 8) y las dificultades que pueden suponer las compatibilidades con el hardware, la compatibilidad de las versiones y los problemas derivados de dependencias, nos obligan a tener alternativas.

En caso de problemas con el hardware consideraríamos utilizar versiones de los programas más actuales. Dado que el hardware utilizado es considerablemente antiguo no deberían existir problemas en esta dirección, salvo quizás, con las librerías gráficas.
En caso de existir, detectar los problemas nos tomaría entre 1 y 3 horas y solucionarlo, nada si prescindimos del componente y unas 10 horas si necesitamos buscarlo y comprarlo.

En el caso de problemas en cuanto a la compatibilidad de algunas versiones optaríamos por compilar directamente versiones compatibles. Vamos a instalar librerías y programas desde los repositorios oficiales de debían lo cual debería de solucionar gran parte de estos problemas.
Si fuese necesario compilar, podría llevar bastante tiempo en función del número de librerías y el tamaño de estas, pero en el peor de los casos no sería superior a 30 horas.

En caso de problemas con las dependencias utilizaríamos las herramientas que nos ofrece el propio dpkg y apt para intentar solucionarlas.
Si fuese imposible tomaríamos la misma dirección que con los problemas de compatibilidad de las versiones. Con un máximo de 30 horas.

En el caso extremo en que no consigamos solucionar los problemas con ninguna de las alternativas propuestas, realizaríamos el proyecto desde cero de nuevo cambiando la versión de debían a debían 7 o 9 en función del tipo de problemas que tengamos. Lo más probable es que para llegar a este extremo, debamos tener graves problema de compatibilidad con el hardware. Tomar esta medida podría llevar a considerar el aplazamiento del turno de lectura.

10. Análisis económico

10.1. Análisis de los roles.

En el siguiente punto trataremos los costes estimados para los recursos materiales, los recursos humanos y los gastos generales derivados directo o indirectamente, en las diferentes fases del proyecto, durante su puesta en producción y durante su vida útil.

Los costes de recursos humanos los calcularemos en función del rol de cada uno y de las tareas que llevará a cabo cada rol.

Rol	*Salario mensual*	*Salario por hora*
Jefe de proyectos	1.687,02€	10,5€
Ingeniero de sistemas	1.253,16	7,8€

Tabla 1: Salario por roles [11]

Cada uno de los roles llevará a cabo un conjunto de tareas con unas horas asignadas en la figura 2. Principalmente el jefe de proyecto hará especialmente gestión y documentación mientras que el ingeniero de sistemas llevará a cabo el resto de tareas.

Rol	*Tarea*	*Horas estimadas*
Jefe de proyecto	Inicio del proyecto	60h
	Gestión del proyecto	65h
	Preparación del entorno	30h

Ingeniero de sistemas	Compilación y creación sistema de ficheros temporal	70h
	Instalación del sistema	90h
	Implementación de programas específicos	80h
	Personalización de la distribución	25h
Total		**420h**

Tabla 2: División de tareas por rol.

10.2. Análisis de presupuesto

Vamos a ver el importe total de los costes directos e indirectos amortizados junto con la partida de contingencia y el presupuesto para imprevistos.

Código	*Tarea*	*Importe*	*Observaciones*
T1	**Inicio del proyecto**		
T1.1	Definición e investigación	420€	
T1.2	Preparación del hardware necesario	210€	
T2	**Gestión del proyecto**		
T2.1	Definición del alcance	157,5€	
T2.2	Planificación temporal	105€	
T2.3	Gestión económica	105€	
T2.4	Reuniones de control	157,5€	
T2.5	Documentación del proyecto	630€	
T3	**Preparación del entorno**		
T3.1	Preparación de particiones	23,4€	
T3.2	Instalación de Ubuntu	78€	
T3.3	Preparación de particiones y fstab	54,6€	
T3.4	Descarga de todos los paquetes necesarios	78€	
T4	**Sistema de ficheros temporal**		
T4.1	Compilación de programas	234€	
T4.2	Instalación de programas	156€	
T5	**Instalación del sistema**		
T5.1	Creación de los sistemas de ficheros virtuales	39€	
T5.2	Compilación e instalación de dpkg	39€	
T5.3	Descarga e instalación de paquetes .deb	78€	
T5.4	Crear ficheros de configuración del sistema	78€	
T5.5	Instalación y creación de usuarios	39€	
T5.6	Descarga e instalación del kernel	78€	
T5.7	Instalación y configuración de Grub	39€	

T5.8	Instalación y configuración de librerías gráficas	117€	
T6	**Implementación de programas específicos**		
T6.1	Compilar e instalar MAME	39€	
T6.2	Configurar MAME	39€	
T6.3	Compilar e instalar EmulationStation	39€	
T6.4	Configurar EmulationStation	78€	
T6.5	Personalizar EmulationStation	312€	
T6.6	Configurar controles	117€	
T7	**Personalización de la distribución**		
T7.1	Configurar autoarranque de EmulationStation	78€	
T7.2	Configurar splash screen Arkadium	78€	
T7.3	Configuración final de Grub	39€	
	Total costes directos	**3.735€**	
	Lenovo ideapad 320	8€	Asumiendo una vida útil de 3 años. Coste del equipo 500€
	Máquina arcade	14€	Asumiendo 10% de la vida útil en el desarrollo.
	Electricidad	30€	Cuota de 45€ mensuales
	Internet	26,6€	Cuota de 40€ mensuales
	Agua	6,6€	Cuota de 10€ mensuales
	Gas	10€	Cuota de 15€ mensuales
	Total costes directos + indirectos	**3.830,2€**	
	Partida de contingencia de un 10 %	383,2€	
	Total costes + contingencia	**4.213,4€**	
	Cambiar procesador (5% prob)	2€	Coste total 40€
	Cambiar placa base (5% prob)	0,75€	Coste total 15€
	Cambiar memoria RAM (5% prob)	1€	Coste total 20€
	Cambiar pantalla (5% prob)	1,5€	Coste total 30€
	Cambiar joysticks (5% prob)	1€	Coste total 20€
	Cambiar botones (5% prob)	1,35€	Coste total 27€
	Cambiar controlador (5% prob)	0,95€	Coste total 19€
	Cambiar disco (5% prob)	2,5€	Coste total 50€
	Total imprevistos	**11,05€**	
	Total	**4.224,45€**	

Tabla 3: Presupuesto general

Como podemos observar la realización completa del proyecto asciende a 4.224,45€ incluyendo todos los gastos posibles. Imprevistos, partida de contingencia, costes directos, indirectos y amortizaciones.

11. Sostenibilidad

11.1. Autoevaluación

A lo largo del grado hemos trabajado competencias transversales relacionadas con la ecología y la sostenibilidad. Participamos en los talleres de “Tecnología per a tothom” [13] reinstalando viejos equipos para darles una segunda vida. Además fuimos instados a participar en el visionado de 3 documentales sobre reciclaje y vertederos tecnológicos en países menos desarrollados socio-económicamente. Documentales con nombre “Comprar, tirar, comprar” y “The e-waste tragedy”. El tercer documental no recuerdo el nombre, pero trata acerca de las mafias y guerrillas que se desarrollan en torno a la extracción del coltán en el Congo.

Todas estas muestras de la realidad que no vemos en nuestra vida convencional te permiten desarrollar un pensamiento crítico respecto a la importancia de los actos de los trabajadores de nuestra profesión, las medidas que podemos tomar, que con un costo muy pequeño para nosotros puede marcar diferencias importantes a miles de kilómetros de distancia.

Aunque no conozco todas las consecuencias sociales y ecológicas de mis actos en términos tecnológicos y de consumo, creo que tengo las herramientas necesarias para comprender el impacto que podemos generar mis compañeros y yo.

11.2. Carácter sostenible del proyecto

Una de las mayores catástrofes ecológicas del siglo XXI es, sin lugar a dudas, la basura tecnológica. Como hemos podido conocer en los últimos años. Algunos países de África, como por ejemplo Ghana, y Asia, como por ejemplo china y en 2018 también Tailandia, según el confidencial [12], se ha convertido en auténtico vertederos de equipos antiguos de todo tipo para los que ha terminado su vida útil.

Según el portal web de la revista Circle [14], a nivel mundial se producen casi 50 millones de toneladas de desechos electrónicos por año. De entre todo ello decenas de minerales preciosos reciclables y varias sustancias altamente contaminantes y tóxicas como el cadmio, los gases refrigerantes de los frigoríficos o el mercurio acaban en vertederos electrónicos.

Una de las formas de combatir esta catástrofe mundial es la de ofrecer una segunda vida útil a productos que están destinados a convertirse en basura. En este caso, darle una vida

útil a equipos viejos que ya no pueden llevar a cabo funciones cotidianas de uso común como por ejemplo navegar por la red u ofimática con las tecnologías y sistemas operativos actuales. Esto incluye procesadores, placas base, memoria RAM, tarjetas gráficas, discos duros y monitores de diferentes tecnologías, tamaños y orígenes.

11.3. Reflexión

Con lo visto en los apartados de sostenibilidad vamos a realizar algunas reflexiones en los diferentes apartados considerables.

11.3.1. Social

A nivel social, facilita el proceso de construcción de una máquina arcade reduciendo los conocimientos técnicos necesarios para llevar a cabo un proyecto de bricolaje de estas características. Incluso puede ser una razón que invite a colaborar a padres y niños uniendo elementos culturales de ambas generaciones.

También puede ayudar a mostrar y popularizar entre las nuevas generaciones, partes o elementos de la cultura digital de generaciones pasadas. Por ejemplo, la gran mayoría de niños de menos de 15 años, nunca, a lo largo de su vida, han visto una máquina arcade ni saben que existieron. De nuevo esto podría beneficiar en el acercamiento de 2 generaciones con una enorme brecha digital.

11.3.2. Ambiental

El impacto ambiental del proyecto se reduce al consumo de los equipos en los cuales se compila y desarrolla. La principal medida que he tomado es la de utilizar un equipo mucho más nuevo y eficiente energéticamente, de la misma arquitectura, para realizar todas la labores de compilación. Reduciendo enormemente el consumo y el tiempo necesario para llevarlo a cabo. Tiempo que de otra forma sería un consumo continuo de energía.

Una vez funcional el sistema temporal que habremos compilado, pasaremos a compilar dpkg e instalar apt para gestionar el resto de paquetes ya pre-compilados, reduciendo enormemente el tiempo de compilación que sería necesario, y la energía necesaria

para ello, y el tiempo del trabajador. Además facilitando el trabajo y reduciendo los conflictos de dependencias.

Si volviese a hacer el proyecto, puede que tuviese equipo mejor y más eficiente pero sin contar con el coste ecológico y económico de la compra de equipo nuevo no creo que pudiese realizarlo mejor en términos ecológicos.

Además invita al reciclaje y reutilización de viejos equipos destinados a la basura. Esto puede generar un impacto positivo en la generación de residuos electrónicos de los países desarrollados.

11.3.3. Económica

Dado al carácter de reutilización de componentes subyacente al proyecto que pretende darles una segunda vida a productos antiguos, puede reducir considerablemente la compra compulsiva de equipos de forma innecesaria, reduciendo los costes ambientales de fabricación masiva y los costes económicos de la adquisición de nuevos productos que de echo en muchas ocasiones no son necesarios.

12. Realización del proyecto

Para llevar a cabo el proyecto seguiremos inicialmente “Linux from Scratch”. Linux from Scratch[15] es un documento desarrollado inicialmente por Gerard Beekmans, al que se han ido sumando miembros con la evolución de las versiones. También cuenta con un grupo de traductores a varios idiomas que hacen el documento mucho más accesible.

Este documento explica como instalar un sistema operativo Linux desde cero, compilando una a una las librerías y creando un sistema de ficheros sobre una partición con la estructura de carpetas y programas característicos de Linux. Llegados a un punto instalaremos el kernel de debían y el sistema gestor de paquetes dpkg (apt) para tener un debían limpio y a partir de este punto comenzaremos a personalizar el sistema hasta obtener *Arkadium*.

12.1. Preparación del entorno

Lo primero que deberíamos hacer será crear las particiones necesarias para llevar a cabo el proyecto.

En nuestro caso necesitaremos 3 particiones. Una de ellas para *Ubuntu*, que será el sistema operativo desde el que trabajemos en las fases iniciales del proyecto contra la partición destianada a *Arkadium*. La segunda partición, será la destinada para el propio sistema *Arkadium*. Será la partición sobre la que trabajaremos la gran mayoría del tiempo. Por último, una partición de swap que puedan utilizar ambos sistemas.

Dado que dispondremos de un sistema operativo Ubuntu que nos proporcionará las herramientas necesarias. Instalaremos primero Ubuntu, junto con GRUB.

12.1.1. Instalación de *Ubuntu*

Llevaremos a cabo una instalación convencional de *Ubuntu*. Durante el proceso de instalación estableceremos una partición ext4 para *Ubuntu* no demasiado grande, entre 20 y 30 GB será más que suficiente. Una partición de swap de 4 GB y nos aseguraremos de dejar al menos 40 GB libres.

Ante la pregunta de si queremos instalar GRUB, debemos contestar afirmativamente e instalarlo en el disco en que estamos instalando *Ubuntu*, generalmente *sda*.

El documento "*Linux from Scratch*" indica en su capítulo "*2.2. Host System Requirements*" que el sistema host, en este caso nuestro *Ubuntu*, debe disponer del software indicado al menos en las versiones siguientes:

- **Bash-3.2** (/bin/sh debería ser un symbolic o hard link a bash)
- **Binutils-2.25** (Versiones más actualizadas que la 2.31.1 no han sido probadas)
- **Bison-2.7** (/usr/bin/yacc debería ser un link a bison o un script que lance bison)
- **Bzip2-1.0.4**
- **Coreutils-6.9**
- **Diffutils-2.8.1**
- **Findutils-4.2.31**
- **Gawk-4.0.1** (/usr/bin/awk debería ser un link a gawk)
- **GCC-4.9** incluyendo el compilador de C++, **g++** (Versiones más actualizadas que la 8.2.0 no han sido probadas)
- **Glibc-2.11** (Versiones más actualizadas que la 2.28 no han sido probadas)
- **Grep-2.5.1a**
- **Gzip-1.3.12**
- **Linux Kernel-3.2**
- **M4-1.4.10**
- **Make-4.0**
- **Patch-2.5.4**
- **Perl-5.8.8**
- **Sed-4.1.5**
- **Tar-1.22**
- **Texinfo-4.7**
- **Xz-5.0.0**

Para verificar que nuestro sistema cumple con los requisitos, podemos ejecutar el siguiente código en un terminal que creará y lanzará un script para comprobar las versiones de cada uno de los programas listados:

```
cat > version-check.sh << "EOF"
      #!/bin/bash
      # Simple script to list version numbers of critical development tools
      export LC_ALL=C
      bash --version | head -n1 | cut -d" " -f2-4
      MYSH=$(readlink -f /bin/sh)
      echo "/bin/sh -> $MYSH"
      echo $MYSH | grep -q bash || echo "ERROR: /bin/sh does not point to bash"
      unset MYSH

      echo -n "Binutils: "; ld --version | head -n1 | cut -d" " -f3-
      bison --version | head -n1

      if [ -h /usr/bin/yacc ]; then
        echo "/usr/bin/yacc -> `readlink -f /usr/bin/yacc`";
      elif [ -x /usr/bin/yacc ]; then
        echo yacc is `/usr/bin/yacc --version | head -n1`
      else
        echo "yacc not found"
      fi

      bzip2 --version 2>&1 < /dev/null | head -n1 | cut -d" " -f1,6-
      echo -n "Coreutils: "; chown --version | head -n1 | cut -d")" -f2
      diff --version | head -n1
      find --version | head -n1
      gawk --version | head -n1

      if [ -h /usr/bin/awk ]; then
        echo "/usr/bin/awk -> `readlink -f /usr/bin/awk`";
      elif [ -x /usr/bin/awk ]; then
        echo awk is `/usr/bin/awk --version | head -n1`
      else
        echo "awk not found"
      fi

      gcc --version | head -n1
      g++ --version | head -n1
      ldd --version | head -n1 | cut -d" " -f2-  # glibc version
      grep --version | head -n1
      gzip --version | head -n1
      cat /proc/version
      m4 --version | head -n1
      make --version | head -n1
      patch --version | head -n1
      echo Perl `perl -V:version`
      sed --version | head -n1
      tar --version | head -n1
      makeinfo --version | head -n1
      xz --version | head -n1

      echo 'int main(){}' > dummy.c && g++ -o dummy dummy.c
      if [ -x dummy ]
        then echo "g++ compilation OK";
        else echo "g++ compilation failed"; fi
      rm -f dummy.c dummy
      EOF

      bash version-check.sh
```

Si alguno de los programas no está instalado o no está en la versión necesaria debemos actualizarlo, generalmente a través de apt.
Si los soft/hard link no están establecidos, es **muy importante** establecerlos dado que son rutas que se utilizarán durante el proceso de compilación e instalación.

12.1.2. Preparación de las particiones

Una vez disponemos de un sistema operativo *Ubuntu* funcional es momento de crear la partición de *Arkadium* y darle formato.

Para ello podemos utilizar la herramienta fdisk incluída en *Ubuntu*.

Asumiendo que el disco es identificado como /dev/sda

```
# sudo fdisk /dev/sda
```

Entramos en el menú de *fdisk* para sda y utilizamos la opción “n” para crear una nueva partición.

Seleccionamos partición primaria, por defecto.

Establecemos el número que nos sugiera *fdisk*, en nuestro caso, 3, por defecto.

Elegimos el sector de inicio por defecto y en el sector d finalización establecemos +40G (40GB).

```
root@debian:/home/citcea# fdisk /dev/sda

Bienvenido a fdisk (util-linux 2.25.2).
Los cambios solo permanecerán en la memoria, hasta que decida escribirlos.
Tenga cuidado antes de utilizar la orden de escritura.


Orden (m para obtener ayuda): n
Tipo de partición
   p   primaria (1 primarias, 1 extendidas, 2 libres)
   l   lógica (la numeración empieza por 5)
Seleccionar (valor predeterminado p):

Se está utilizando la respuesta predeterminada p.
Número de partición (3,4, valor predeterminado 3):
Primer sector (40136704-41943039, valor predeterminado 40136704):
Último sector, +sectores o +tamaño{K,M,G,T,P} (40136704-40138749, valor predeterminado 40138749): +40G
```

Por últimos elegimos la opción w para escribir la tabla de particiones en el disco.

Con esto deberíamos de tener una tabla de particiones. Es posible que se produzca un error de lectura de la tabla porque el sistema operativo actual está utilizando el disco. Eso no debería de suponer ningún tipo de problema.

Por regla general, y para prevenir algún tipo de problema, sería recomendable reiniciar el equipo para asegurarnos que la tabla de particiones se está leyendo correctamente.

Una vez con el sistema funcionando y la tabla de particiones correctamente establecida debemos establecer un nuevo sistema de ficheros para la nueva partición (en nuestro caso /dev/sda2), para ello utilizaremos el comando *mkfs*.

```
# sudo mkfs –v –t ext4 /dev/sda2
```

mkfs –v: muestra por pantalla lo que se lleva a cabo
mkfs –t ext4: se creará un sistema de ficheros de tipo ext4.

12.1.3. Configuración de variables y fstab

En términos generales este es un proceso largo, especialmente si no disponemos de un equipo potente sobre el que trabajar en el momento de compilar. Dado que, necesitaremos montar la partición del sistema operativo destino, si vamos a hacerlo en varias sesiones es recomendable modificar el fichero *fstab* para que monte automáticamente las particiones en el proceso de inicio del sistema *Ubuntu* que estamos utilizando.

Lo primero que debemos tener en cuenta es que, para facilitar el uso de rutas, estableceremos una variable de entorno LFS como *"/mnt/lfs"* que es la ruta donde montaremos la nueva partición.

```
# export LFS=/mnt/lfs
```

Posteriormente y utilizando la variable LFS vamos a crear el directorio "*/mnt/lfs*" y montamos la nueva partición sobre el nuevo directorio (en nuestro caso /dev/sda2).

```
# sudo mkdir -pv $LFS
# sudo mount -v -t ext4 /dev/sda2 $LFS
```

mkdir –p: Crea directorios padres si es necesario.
mkdir –v: Muestra la información de lo que se hace.
mount –v: Muestra la información de lo que se hace.
mount –t ext4: la partición que va a montarse es de tipo "ext4".

Nota opcional: Si reiniciamos o apagamos el equipo por alguna razón, deberemos de volver a montar la partición. Si tenemos intención de apagar o reiniciar el equipo varias veces, lo mejor es modificar el fichero *"/etc/fstab"* añadiendo una línea indicando la partición a montar (en nuestro caso /dev/sda2).

```
/dev/sda2       /mnt/lfs      ext4     defaults        1       1
```

Nota opcional: Si disponemos de una partición de swap, lo que es altamente recomendable, pero por alguna razón no se está utilizando o no estamos seguros, podemos garantizar su uso utilizando *swapon* sobre la partición de swap (por ejemplo /dev/sda3).

```
# /sbin/swapon -v /dev/sda3
```

12.1.4. Descarga de los paquetes necesarios

Para compilar todos los programas básicos que forman el sistema Linux, lo primero que debemos hacer descargarlos. El propio documento "*Linux from Scratch*" proporciona una lista de enlaces en los cuales obtener todos los códigos fuente de todos los programas necesarios para crear un sistema básico. De hecho, proporciona también varios programas que no necesitamos dado que seguiremos el documento "*Linux from Scratch*" solo hasta cierto punto.

Antes de descargar todos los códigos fuente crearemos un directorio donde almacenarlos y le daremos permisos para que todos los usuarios puedan acceder y escribir.

```
# sudo mkdir -v $LFS/sources
# sudo chmod -v a+wt $LFS/sources
```

chmod –v: Muestra por pantalla las acciones que se llevan a cabo.
chmod a: Se aplica a todos los usuarios.
chmod +w: proporciona permiso de escritura.
chmod +t: utiliza sticky bit. Solo el propietario del directorio puede renombrar o eliminar documentos.

Ya tenemos todo listo para descargar los códigos. Para ello, vamos a crear un documento llamado *wget-list* incluyendo todos los enlaces de todos los códigos que vamos a bajarnos [ANEXO]. Lo guardamos en un directorio en el que tengamos permisos y utilizamos el siguiente comando para realizar las descargas.

```
# sudo wget --input-file=wget-list --continue --directory-prefix=$LFS/sources
```

wget –input-file: Utiliza el fichero indicado para obtener los enlaces de descarga
wget –continue: Continua una descarga si por alguna razón no ha terminado.
wget –directory-prefix: Directorio donde se descargarán los ficheros o directorios.

12.2. Compilación y creación de un sistema temporal

12.2.1. Preparación de entorno para compilar

Para llevar a cabo la compilación e instalación de todos los programas necesarios seguiremos al pie de la letra el capítulo.

Crearemos un nuevo directorio en la raíz de la nueva partición, llamado *tools*, en el que guardaremos los programas una vez compilados, para ello utilizaremos el comando *mkdir*.

```
# sudo mkdir –v $LFS/tools
```

El próximo paso es crear un enlace */tools* en el sistema anfitrión. Este apuntará al directorio que acabamos de crear en la partición LFS. Ejecuta este comando también como root.

```
# sudo ln -sv $LFS/tools /
```

El enlace simbólico creado posibilita que el conjunto de herramientas se compile siempre en referencia a */tools*, de forma que el compilador, ensamblador y enlazador funcionarán ahora (que todavía estamos utilizando algunas herramientas del sistema anfitrión) y más adelante (cuando "cambiemos la raíz" a la partición LFS).

A continuación, vamos a crear el grupo y el usuario "*lfs*". Utilizaremos los comandos *groupadd* y *useradd*.

```
# sudo groupadd lfs
# sudo useradd -s /bin/bash -g lfs -m -k /dev/null lfs
```

useradd –s /bin/bash: Inidica el Shell establecido para ese usuario.

useradd –g: Indica el grupo por defecto del usuario.

useradd –m: Indica que se creará un directorio home para este usuario.

useradd –k /dev/null: Indica el directorio base, con el que se crea el home.

Ahora debemos establecer una contraseña para el nuevo usuario con el comando *passwd*.

```
# sudo passwd lfs
```

De ahora en adelante trabajaremos con el usuario *lfs*, de modo que le daremos la propiedad del directorio *source* y *tools*.

```
# sudo chown –v lfs $LFS/tools
# sudo su chown –v lfs $LFS/sources
```

chown –v: muestra los cambios realizados.

Ahora, que ambas carpetas son propiedad del usuario *lfs*, vamos a cambiar de usuario.

```
# su – lfs
```

El parámetro "-" indica que se inicie el nuevo usuario en una login Shell en vez de una non-login Shell, que es el comportamiento por defecto.

Ahora vamos a establecer un buen entorno de trabajo mediante la creación de dos nuevos ficheros de inicio para el intérprete de comandos bash. Estando en el sistema como usuario lfs, ejecuta los siguientes comandos para crear un *.bash_profile* nuevo:

```
cat > ~/.bash_profile << "EOF"
exec env -i HOME=$HOME TERM=$TERM PS1='\u:\w\$ ' /bin/bash
EOF
```

Cuando entras como usuario *lfs* el intérprete de comandos inicial es *login shell* que lee el */etc/profile* de tu anfitrión (que posiblemente contenga algunos ajustes de variables de entorno) y luego lee *.bash_profile*. El comando ***exec env -i.../bin/bash*** del fichero *.bash_profile* sustituye el intérprete de comandos en ejecución por uno nuevo con un entorno completamente vacío, excepto por las variables *HOME, TERM y PS1*. Esto asegura que en el entorno de construcción no aparezcan variables de entorno indeseadas o dañinas procedentes del sistema anfitrión. La técnica aquí usada consigue el objetivo de asegurar un entorno limpio.

La nueva instancia del intérprete comandos es una *non-login shell* que no lee los ficheros */etc/profile o .bash_profile*, en su lugar lee el fichero *.bashrc*. Crea ahora el fichero *.bashrc*:

```
cat > ~/.bashrc << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
LFS_TGT=$(uname -m)-lfs-linux-gnu
PATH=/tools/bin:/bin:/usr/bin
export LFS LC_ALL LFS_TGT PATH
EOF
```

El comando *set +h* desactiva la función de tablas de dispersión (hash) de bash. Normalmente, esta función es muy útil: bash usa una tabla de dispersión para recordar la ruta completa de los ejecutables, evitando búsquedas reiteradas en el PATH para encontrar el mismo binario. Sin embargo, las nuevas herramientas deberían utilizarse a medida que son instaladas. Al desactivar esta característica, el intérprete de comandos siempre buscará en el

PATH cuando deba ejecutarse un programa. Por tanto, el intérprete de comandos encontrará las herramientas recién compiladas en $LFS/tools tan pronto como estén disponibles, sin recordar una anterior versión del mismo programa en una ubicación diferente.

Establecer la máscara de creación de ficheros (umask) a 022 asegura que los ficheros y directorios de nueva creación sólo pueden ser escritos por su propietario, pero son legibles y ejecutables por cualquiera (asumiendo que los modos por defecto son usados por la llamada open(2) del sistema, los nuevos ficheros tendrán permisos 644 y los directorios 755).

La variable LFS debe establecerse al punto de montaje utilizado, en nuestro caso */mnt/lfs*.

La variable LC_ALL controla la localización de ciertos programas, haciendo que sus mensajes sigan las convenciones para un determinado país. Si el sistema anfitrión utiliza una versión de Glibc anterior a la 2.2.4, tener LC_ALL establecida a algo diferente a “POSIX” o “C” puede causar problemas si sales del entorno *chroot* e intentas regresar más tarde. Establecer LC_ALL a “POSIX” o “C” (ambos son equivalentes) asegura que todo funcionará como se espera dentro del entorno *chroot*.

La variable LFS_TGT establece la descripción de una máquina no por defecto, pero compatible para el cross-compiling. Básicamente establece el *target* del compilador.

Al añadir “*/tools/bin*” al principio del PATH, todos los programas que compilemos e instalemos, serán inmediatamente detectados por el intérprete de comandos tras su instalación. Esto, combinado con la desactivación de las tablas de dispersión, limita el riesgo de utilizar los antiguos programas del anfitrión cuando dichos programas ya están disponibles en nuestro entorno temporal.

Finalmente, para tener el entorno preparado por completo para construir las herramientas temporales, carga el perfil de usuario recién creado:

```
# source ~/.bash_profile
```

12.2.2. Compilación e instalación de programas

Una vez llegados a este punto ya tenemos todo listo para empezar a compilar. La gran mayoría de procesadores desde hace décadas disponen de más de un núcleo o hilos de ejecución, podemos indicar al compilador cuantos núcleos o hilos de ejecución queremos que utilice estableciendo la variable de entorno *MAKEFLAGS:*

```
#  export MAKEFLAGS='-j 2'
```

En este caso de ejemplo, establecemos el uso de 2 núcleos o hilos de ejecución durante la compilación.

Es importante remarcar que el proceso de compilación puede extenderse durante muchas horas o incluso días si se utilizan procesadores muy antiguos y poco potentes.

Llegados a este punto es importante asegurarse de que el host cumple con todos los requisitos mencionados en el punto *instalación de Ubuntu*.

Binutils-2.31.1 – paso 1

Binutils se compilará en 2 fases. Es importante que esta primera fase se la primera en compilarse porque *Glibc* y *GCC* realizan varios test con el asembler y linker disponibles para saber qué características activar.

Lo primero será descomprimir el fichero con el código.

```
# tar -xvf binutils-2.31.1.tar.xz
```

tar –x: Indica que vamos a descomprimir.

tar –v: Muestra los ficheros que se están descomprimiendo.

tar –f: Indicamos un fichero específico.

Esto creará una carpeta (binutils – 2.26) en el directorio *sources* con los ficheros descomprimidos.

Accedemos a la carpeta, creamos un directorio *build* para contener los ficheros generados durante la compilación y accedemos a él.

```
# cd binutils-2.31.1
# mkdir build
# cd build
```

Ahora vamos a configurar *binutils* para compilarlo estableciendo las opciones correctas.

```
../configure --prefix=/tools      \
       --with-sysroot=$LFS        \
       --with-lib-path=/tools/lib \
       --target=$LFS_TGT          \
       --disable-nls              \
       --disable-werror
```

--prefix=/tools

Indica al script de configuración que se prepare para instalar *binutils* en el directorio "*/tools*"

--with-sysroot=$LFS

Para compilación cruzada, indica al sistema de compilación que mira en el directorio $LFS para ver las librerías de sistema necesarias para el target.

--with-lib-path=/tools/lib

Especifica qué ruta deberá consultar para las librerías.

--target=$LFS_TGT

Esto dirá al script de configuración los ajustes de enlace y compilación cruzada.

--disable-nls

Esto desactiva la internacionalización.

--disable-werror

Evita que la compilación se detenga si se producen warnings.

Una vez realizada la configuración compilaremos utilizando el comando *make.*

make

Este proceso puede tardar bastante.
Al finalizar procederemos a instalarlo, pero tenemos que tener en cuenta nuestra arquitectura. Si estamos compilando en una máquina de 64 bits, crearemos un soft link para garantizar el correcto uso de las rutas. Para ello ejecutaremos el siguiente código en el terminal.

```
case $(uname -m) in
       x86_64) mkdir -v /tools/lib && ln -sv lib /tools/lib64 ;;
esac
```

Por último, usamos el comando *make install* para llevar a cabo la instalación en las rutas establecidas.

```
# make install
```

Tras esto, veremos que se ha creado en "$*LFS/tools*" una estructura de carpetas que nos suena si estamos familiarizados con sistemas operativos GNU/Linux. *bin, include, lib, share, x86_64-pc-linux-gnu.*

Podemos eliminar el directorio binutils-2.26 para reducir el espacio ocupado

Gcc-8.2.0 – paso 1

Igual que con *binutils*, *GCC* lo compilaremos en 2 partes.

Lo primero será descomprimir *GCC* usando el comando *tar*.

```
# tar -xvf gcc-8.2.0.tar.xz
```

GCC necesita para llevar a cabo la compilación, los paquetes *mpfr, gmp y mpc*. Para ellos vamos a descomprimirlos dentro de la misma carpeta en que hemos descomprimido *GCC* y a renombrarlas para que las rutas sean correctas.

```
# cd gcc-8.2.0
# tar -xf ../mpfr-4.0.1.tar.xz
# mv -v mpfr-4.0.1 mpfr
# tar -xf ../gmp-6.1.2.tar.xz
# mv -v gmp-6.1.2 gmp
# tar -xf ../mpc-1.1.0.tar.gz
# mv -v mpc-1.1.0 mpc
```

Para evitar que se produzcan problemas con las rutas durante la compilación debemos modificar en enlazador dinámico por defecto para que se utilice el instalado en "*/tools*". Para llevar a cabo ese cambio copiaremos esto en el terminal.

```
      for file in gcc/config/{linux,i386/linux{,64}}.h
do
  cp -uv $file{,.orig}
  sed -e 's@/lib\(64\)\?\(32\)\?/ld@/tools&@g' \
      -e 's@/usr@/tools@g' $file.orig > $file
  echo '
#undef STANDARD_STARTFILE_PREFIX_1
#undef STANDARD_STARTFILE_PREFIX_2
#define STANDARD_STARTFILE_PREFIX_1 "/tools/lib/"
```

Por último, antes de compilar debemos establecer la ruta de "*lib*" si nuestro equipo es de 64 bits. Lo haremos utilizando el siguiente código en la terminal.

```
      case $(uname -m) in
  x86_64)
    sed -e '/m64=/s/lib64/lib/' \
        -i.orig gcc/config/i386/t-linux64
 ..
```

Como ya hemos hecho con *binutils*, vamos a generar un directorio *build* para contener todos los ficheros producidos durante la compilación.

```
# mkdir -v build
# cd build
```

Y ahora vamos a configurar la compilación con *configure*

```
../configure                                       \
      --target=$LFS_TGT                            \
      --prefix=/tools                              \
      --with-glibc-version=2.11                    \
      --with-sysroot=$LFS                          \
      --with-newlib                                \
      --without-headers                            \
      --with-local-prefix=/tools                   \
      --with-native-system-header-dir=/tools/include \
      --disable-nls                                \
      --disable-shared                             \
      --disable-multilib                           \
      --disable-decimal-float                      \
      --disable-threads                            \
      --disable-libatomic                          \
      --disable-libgomp                            \
      --disable-libmpx                             \
      --disable-libquadmath                        \
      --disable-libssp                             \
      --disable-libvtv                             \
      --disable-libstdcxx                          \
      --enable-languages=c,c++
```

--with-newlib

Esta opción impide que se utilice código que requiera soporte de *libc* dado que aún no hemos compilado *libgcc*.

--without-headers

Para compilación cruzada *GCC* requiere cabeceras compatibles con el sistema target. En nuestro caso no es así, de modo que esta opción evita que las busque.

--with-local-prefix=/tools

Local-prefix es la ubicación donde *GCC* buscará los *include files*. Por defecto esta ubicación es */usr/local*. De esta forma buscará */usr/local* dentro de la raíz de nuestro nuevo sistema.

--with-native-system-header-dir=/tools/include

GCC busca las cebeceras del sistema, por defecto, en *"/usr/include" con el cambio de root,* se convertirá en *"$LFS/usr/include"* en vez de "*$LFS/tools/include*" que sería lo que nos interesaría. Con esto establecemos la ruta de búsqueda de librerías del sistema en "*/tools/include*".

--disable-shared

Esta opción indica a GCC que debe enlazar sus librerías de forma estática. Evitando así problemas con el host.

--disable-decimal-float, --disable-threads, --disable-libatomic, --disable-libgomp, --disable-libmpx, --disable-libquadmath, --disable-libssp, --disable-libvtv, --disable-libstdcxx

Estas opciones deshabilitan: *decimal floating point extension, threading, libatomic, libgomp, libmpx, libquadmath, libssp, libvtv, y C++ standard library.*

Estas opciones se utilizan en compilación cruzada, en nuestro caso pueden producir problemas así que vamos a deshabilitarlas.

--disable-multilib

En arquitectura x86_64, LFS no tiene soporte para *multilib* aún.

--enable-languages=c,c++

Esta opción garantiza que solo se compilarán los compiladores de C y C++. Son los únicos lenguajes que necesitamos por ahora.

Ahora toca compilar, este proceso puede tardar bastante.

```
#make
#make install
```

Una vez finalizado el proceso ya tendremos listo nuestro propio compilador.

Linux 4.18.5 API Headers

Ahora vamos a instalar algunas de las cabeceras de la API de Linux-4.18 necesarias. Como en los anteriores descomprimimos los datos y en este caso ejecutaremos directamente:

```
# tar -xvf linux-4.18.5.tar.xz
# cd linux-4.18.5
# make mrproper
# make INSTALL_HDR_PATH=dest headers_install
# cp –rv dest/include/* /tools/include
```

Con esto deberíamos tener las cabeceras correctamente instaladas en "*/tools/include*" del nuevo sistema operativo.

Glibc-2.28

Igual que en las compilaciones anteriores descomprimimos el paquete, accedemos a la carpeta y creamos una carpeta *build* que contendrá los ficheros generados durante la compilación.

```
# tar -xvf glibc-2.28.tar.xz
# cd glibc-2.28
# mkdir build
# cd build
```

Ahora pasaremos a configurar el script de compilación.

```
../configure                                \
            --prefix=/tools                 \
            --host=$LFS_TGT                 \
            --build=$(../scripts/config.guess)  \
            --enable-kernel=3.2             \
            --with-headers=/tools/include   \
            libc_cv_forced_unwind=yes       \
            libc_cv_c_cleanup=yes
```

--host=$LFS_TGT, --build=$(../scripts/config.guess)

El efecto combinado de estas 2 opciones es que el sistema de construcción de *glibc* se configura a sí mismo para la compilación cruzada usando el compilador y enlazador de /tools

--enable-kernel=3.2

Indica que se compilará la librería con soporte para linux-3.2 y superior.

--with-headers=/tools/include

Indica que utilice las cabeceras disponibles en "*/tools/include*" que acabamos de instalar.

libc_cv_forced_unwind=yes

El enlazador instalado durante la compilación e instalación de *binutils* no puede utilizarse hasta instalar *glibc*, por ello, el test de configuración fallará. Esta opción indica que todo está bien sin necesidad de hacer el test

libc_cv_c_cleanup=yes

Similar al caso anterior, ignoraremos un test que sabemos que dará error y continuaremos con el proceso.

Es posible que aparezcan mensajes de "*warning*" como los siguientes que no deberían preocuparnos.

```
configure: WARNING:
*** These auxiliary programs are missing or
*** incompatible versions: msgfmt
*** some features will be disabled.
*** Check the INSTALL file for required versions.
```

Con todo listo vamos a proceder a compilar e instalar *glibc*.

```
# make
# make install
```

IMPORTANTE

Llegados a este punto es importante que verifiquemos que todo lo que hemos instalado está funcionando correctamente. Compilaremos un pequeño programa simple para verificarlo. Ejecutaremos el siguiente código en la terminal.

```
echo 'int main(){}' > dummy.c
$LFS_TGT-gcc dummy.c
readelf -l a.out | grep ': /tools'
```

Si todo está funcionando correctamente nos aparecerá el siguiente mensaje:

```
[Requesting program interpreter: /tools/lib64/ld-linux-x86-64.so.2]
```

Si no obtenemos el mensaje esperado es recomendable volver a empezar el proceso desde el principio. En caso de que todo vaya bien, eliminaremos el pequeño programa y su fichero compilado y continuaremos adelante.

```
# rm -v dummy.c a.out
```

Libstdc++ (de gcc-8.2.0)

Libstdc++ es una parte de *gcc* así que necesitamos de nuevo descomprimir *gcc*, acceder y crear de nuevo la carpeta build.

```
# tar -xvf gcc-8.2.0.tar.xz
# cd gcc-8.2.0
# mkdir build
# cd build
```

Vamos a configurar el script de compilación

```
../libstdc++-v3/configure               \
      --host=$LFS_TGT                   \
      --prefix=/tools                   \
      --disable-multilib                \
      --disable-nls                     \
      --disable-libstdcxx-threads       \
```

```
      --disable-libstdcxx-pch            \
      --with-gxx-include-dir=/tools/$LFS_TGT/include/c++/8.2.0
```

--host=...

Usa el compilador que acabamos de compilar e instalar en vez del que hay en "*/usr/bin*"

--disable-libstdcxx-threads

Dado que no hemos compilado aún las librerías de hilos de C, tampoco podemos contruir la de C++.

--disable-libstdcxx-pch

Evita que se instalen ficheros precompilados que no son necesarios para nosotros.

--with-gxx-include-dir=/tools/$LFS_TGT/include/c++/8.2.0

Indicación explícita de la ruta donde se buscarán los ficheros include por parte del compilador de C++

Binutils-2.31.1 – paso 2

Vamos a llevar a cabo la segunda, y última, parte de la compilación de *binutils*.

Descomprimimos de nuevo el paquete y de nuevo creamos el directorio build.

```
# tar -xvf binutils-2.31.1.tar.xz
# cd binutils-2.31.1
# mkdir build
# cd build
```

Configuramos el script de compilación.

```
      CC=$LFS_TGT-gcc                  \
      AR=$LFS_TGT-ar                   \
      RANLIB=$LFS_TGT-ranlib           \
      ../configure                     \
            --prefix=/tools            \
            --disable-nls              \
            --disable-werror           \
            --with-lib-path=/tools/lib \
            --with-sysroot
```

CC=$LFS_TGT-gcc AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib

Dado que es una compilación nativa, establecer los valores de estas variables nos garantiza el uso de las herramientas nativas en vez de las del host.

--with-lib-path=/tools/lib

Esta opción indica la ruta en que buscar las librerías durante la compilación de *binutils*.

--with-sysroot

Habilita características de *sysroot* que permite acceso a objetos compartidos. Sin esta opción podríamos recibir opción en algunos hosts.

Una vez configurado procedemos a compilar e instalar.

```
# make
# make install
```

Con esto finalizamos con *binutils* y vamos a por la segunda parte de *gcc*

Gcc-8.2.0 – paso 2

En el primer paso, algunas de las cabeceras no existían, particularmente *límitis.h* y por tanto *gcc* no está completamente instalado, solo parcialmente. Esta vez necesitaremos que la cabecera se encuentre completa de forma que la crearemos con el siguiente código después de descomprimir gcc y acceder al directorio:

```
# tar -xvf gcc-8.2.0.tar.xz
# cd gcc-8.2.0
```

```
cat gcc/limitx.h gcc/glimits.h gcc/limity.h > \
  `dirname $($LFS_TGT-gcc -print-libgcc-file-name)`/include-
fixed/limits.h
```

De nuevo cambiamos las rutas por defecto del enlazador dinámico para usar el instalado en "/*tools"* con el siguiente código.

```
for file in gcc/config/{linux,i386/linux{,64}}.h
do
  cp -uv $file{,.orig}
  sed -e 's@/lib\(64\)\?\(32\)\?/ld@/tools&@g' \
      -e 's@/usr@/tools@g' $file.orig > $file
  echo '
#undef STANDARD_STARTFILE_PREFIX_1
#undef STANDARD_STARTFILE_PREFIX_2
#define STANDARD_STARTFILE_PREFIX_1 "/tools/lib/"
#define STANDARD_STARTFILE_PREFIX_2 ""' >> $file
  touch $file.orig
done
```

Cambiamos el nombre del directorio *lib* de 64 bits a *lib* si usamos una máquina de 64 bits con el siguiente código:

```
case $(uname -m) in
  x86_64)
    sed -e '/m64=/s/lib64/lib/' \
        -i.orig gcc/config/i386/t-linux64
  ;;
esac
```

Como en la primera parte necesitaremos descomprimir y modificar el nombre de los directorios de *mpfr, gmp y mpc*.

```
# tar -xf ../mpfr-4.0.1.tar.xz
# mv -v mpfr-4.0.1 mpfr
# tar -xf ../gmp-6.1.2.tar.xz
# mv -v gmp-6.1.2 gmp
# tar -xf ../mpc-1.1.0.tar.gz
# mv -v mpc-1.1.0 mpc
```

Ya tenemos listo todo lo necesario para empezar a configurar la compilación así que creamos el directorio build:

```
# mkdir build
# cd build
```

Y lanzamos el comando de configuración del script:

```
CC=$LFS_TGT-gcc                                    \
CXX=$LFS_TGT-g++                                   \
AR=$LFS_TGT-ar                                     \
RANLIB=$LFS_TGT-ranlib                             \
../configure                                       \
      --prefix=/tools                              \
      --with-local-prefix=/tools                   \
      --with-native-system-header-dir=/tools/include \
      --enable-languages=c,c++                     \
      --disable-libstdcxx-pch                      \
      --disable-multilib                           \
      --disable-bootstrap                          \
      --disable-libgomp
```

--enable-languages=c,c++
Esta opción garantiza que ambos compiladores, C y C++ están compilados.

--disable-libstdcxx-pch
Indicia que no vamos a utilizar las cabeceras precompiladas para libstdc++.

--disable-bootstrap
Por defecto, se suele hacer una compilación *bootstrap* que compila varias veces gcc para garantizar que todo se ha realizado correctamente. Si hemos seguido los pasos correctamente esto no debería ser necesario.

Ya tenemos todo listo para compilar e instalar:

```
# make
# make install
```

Por último, creamos un *softlink* a *gcc.* Muchos scripts utilizan *cc* en vez de *gcc*

```
# ln -sv gcc /tools/bin/cc
```

IMPORTANTE

Llegados a este punto es importante que verifiquemos de nuevo que todo lo que hemos instalado está funcionando correctamente. Compilaremos un pequeño programa simple para verificarlo. Ejecutaremos el siguiente código en la terminal.

```
echo 'int main(){}' > dummy.c
cc dummy.c
readelf -l a.out | grep ': /tools'
```

Si todo está funcionando correctamente nos aparecerá el siguiente mensaje:

```
[Requesting program interpreter: /tools/lib64/ld-linux-x86-64.so.2]
```

Si no obtenemos el mensaje esperado es recomendable volver a empezar el proceso desde el principio. En caso de que todo vaya bien, eliminaremos el pequeño programa y su fichero compilado y continuaremos adelante.

```
# rm -v dummy.c a.out
```

Tcl-8.6.8

Este y los próximos 2 programas que instalaremos (*Expect* y *DejaGNU*) son para dar soporte a los tests que haremos sobre *gcc* y *binutils*. Puede parecer excesivo instalar 3 programas para hacer tests pero, es razonable querer asegurarse de que las herramientas más importantes del sistema funciona correctamente.

Empezaremos, como siempre, descomprimiendo el paquete y accediendo al directorio.

```
# tar -xvf tcl8.6.8-src.tar.gz
# cd tcl8.6.8
```

En vez de crear *build*, esta vez accederemos al directorio *unix* y lanzaremos la configuración del script.

```
# cd unix
# ./configure --prefix=/tools
```

Una vez listo, compilamos

```
# make
```

Ahora que ya tenemos el programa compilado vamos a lanzar el test

```
# TZ=UTC make test
```

Es posible que, en función del host, se den algunos fallos no críticos, no deberían de preocuparnos.

Si todo ha ido bien, instalaremos el programa.

```
# make install
```

Ahora vamos a hacer escribible una librería instalada para poder eliminar algunos símbolos más tarde.

```
# chmod -v u+w /tools/lib/libtcl8.6.so
```

Vamos a instalar las cabeceras de *Tcl*. El próximo paquete, *Expect* lo necesitará para compilarse.

```
# make install-private-headers
```

Y por último hacemos un enlace simbólico

```
# ln -sv tclsh8.6 /tools/bin/tclsh
```

Expect-5.45.4

De nuevo, el primer paso será descomprimir el paquete.

```
# tar -xvf expect5.45.4.tar.gz
# cd expect5.45.4
```

Lo primero que haremos, será indicar al script de configure que debe usar *"/bin/stty"* en vez de *"/usr/local/bin/stty"* que es lo que encontraría probablemente en nuestro sistema.

```
cp -v configure{,.orig}
     sed 's:/usr/local/bin:/bin:' configure.orig > configure
```

Ahora vamos a configurar el script de compilación de *expect*.

```
./configure --prefix=/tools            \
            --with-tcl=/tools/lib      \
            --with-tclinclude=/tools/include
```

--with-tcl=/tools/lib

Esta opción garantiza que el script de configuración localiza el programa Tcl instalado en "*/tools/lib*" en vez de un posible programa Tcl instalado en el host.

--with-tclinclude=/tools/include
Indica explícitamente donde encontrar las cabeceras de *Tcl*.

Ya tenemos todo listo para pasar a compilar *Expect,* realizar los test e instalarlo.

```
# make
# make test
# make SCRIPTS=”” install
```

SCRIPTS=””
Evita la instalación de scripts de *Expect* que no son necesarios.

No debería preocuparnos que se produzcan varios fallos en los tests, son normales en función del host.

DejaGNU-1.6.1

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos y realizamos los test.

```
# tar -xvf dejagnu-1.6.1.tar.gz
# cd dejagnu-1.6.1
# ./configure --prefix=/tools
# make install
# make check
```

M4-1.4.18

De nuevo, empezamos por descomprimir y acceder al nuevo directorio.

```
# tar -xvf m4-1.4.18.tar.xz
# cd m4-1.4.18
```

Glibc-2.28 puede generar algunos problemas así que vamos a solucionarlos para evitar que puedan darse.

```
sed -i 's/IO_ftrylockfile/IO_EOF_SEEN/' lib/*.c
echo "#define _IO_IN_BACKUP 0x100" >> lib/stdio-impl.h
```

Configuramos el script de compilación, compilamos, hacemos los tests e instalamos.

```
# ./configure --prefix=/tools
# make
# make check
# make install
```

Ncurses-6.1

Ncurses contiene las librerías para gestionar los caracteres que se muestran por pantalla.

Empezaremos por descomprimir el paquete y acceder a la carpeta.

```
# tar -xvf ncurses-6.1.tar.gz
# cd ncurses-6.1
```

Primero nos aseguraremos de que encuentra *gawk* durante la compilación con el comando:

```
# sed -i s/mawk// configure
```

Ya tenemos todo listo para configurar el script de compilación.

```
./configure --prefix=/tools \
      --with-shared   \
      --without-debug \
      --without-ada   \
      --enable-widec  \
      --enable-overwrite
```

--without-ada
Esta opción garantiza que *Ncurses* no compile con soporte para compilador de *Ada.* Que, aunque pueda estar disponible en el *host* no lo estará en nuestro sistema.

--enable-overwrite
Indica a *Ncurses* que debe instalar sus cabeceras en "*/tools/include*" en vez de en "*/tools/include/ncurses*". De esta forma nos aseguramos de que otros paquetes puedan encontrar sus cabeceras sin problemas.

--enable-widec
Establece que se compilarán la librerías "*wide-character*" en vez de las normales.

Ya estamos listos para compilar e instalar el paquete.

```
# make
# make install
```

Bash-4.4.18

Vamos a instalar ahora *Bash.* Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos los tests y por último instalamos.

```
# tar -xvf bash-4.4.18.tar.gz
# cd bash-4.4.18
# ./configure --prefix=/tools --without-bash-malloc
# make
#make test
#make install
```

Por último, creamos un enlace de sh a bash.

```
# ln -sv bash /tools/bin/sh
```

Bison-3.0.5

Bison es un paquete que contiene un *"parser generator".*

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar -xvf bison-3.0.5.tar.xz
# cd bison-3.0.5
# ./configure --prefix=/tools
# make
# make check
# make install
```

Es completamente normal que el comando *"make check"* devuelva un error.

Bzip2-1.0.6

Este paquete contiene programas de compresión y descompresión de ficheros.

Descomprimimos, accedemos al directorio, compilamos e instalamos, en este caso la configuración se realiza durante la instalación, directamente.

```
# tar -xvf bzip2-1.0.6.tar.gz
# cd bzip2-1.0.6
# make
# make PREFIX=/tools install
```

Coreutils-8.30

Contiene herramientas para mostrar y establecer características básicas del sistema.

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar -xvf coreutils-8.30.tar.xz
# cd coreutils-8.30
# ./configure --prefix=/tools --enable-install-program=hostname
# make
# make RUN_EXPENSIVE_TESTS=yes check
# make install
```

Diffutils-3.6

Contiene programas que muestran diferencias entre ficheros o directorios.

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar -xvf diffutils-3.6.tar.xz
# cd diffutils-3.6
# ./configure --prefix=/tools
# make
# make check
# make install
```

File-5.34

Contiene herramientas para determinar el tipo de ficheros.

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar -xvf file-5.34.tar.gz
# cd file-5.34
# ./configure --prefix=/tools
# make
# make check
# make install
```

Findutils-4.6.0

Contiene programas para encontrar ficheros. Estos programas proveen de búsquedas recursivas a través de árboles.

Descomprimimos y accedemos al fichero.

```
# tar -xvf findutils-4.6.0.tar.gz
# cd findutils-4.6.0
```

Como ya hemos hecho en paquetes anteriores vamos a solucionar algunos problemas conocidos de glibc con el siguiente código en el terminal.

```
sed -i 's/IO_ftrylockfile/IO_EOF_SEEN/' gl/lib/*.c
sed -i '/unistd/a #include <sys/sysmacros.h>'
gl/lib/mountlist.c
echo "#define _IO_IN_BACKUP 0x100" >> gl/lib/stdio-impl.h
```

Una vez solucionado esto, configuramos el script de configuración, compilaremos, haremos los test e instalaremos.

```
# ./configure --prefix=/tools
# make
# make check
# make install
```

Gawk-4.2.1

Este paquete contiene programas para manipular ficheros de texto.

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar -xvf gawk-4.2.1.tar.xz
# cd gawk-4.2.1
# ./configure --prefix=/tools
# make
# make check
# make install
```

En función del host es posible que los tests den errores críticos, no es un problema.

Gettext-0.19.8.1

Contiene herramientas de internacionalización y localización. Permite que los programas se compilen con *NLS* (*Native Languaje Support*), permitiendo que lo mensajes de output sean en el lenguaje nativo del usuario.

En este caso el proceso va a diferir un poco respecto a los anteriores.
Empezamos por descomprimir y acceder al directorio.

```
# tar -xvf gettext-0.19.8.1.tar.xz
# cd gettext-0.19.8.1
```

Prepararemos el script de compilación dentro del directorio *"gettext-tools"*

```
# cd gettext-tools
# EMACS="no" ./configure --prefix=/tools --disable-shared
```

EMACS="no"
Esta opción evita que se busque un *EMACS* instalado.

--disable-shared
Indica que no instalaremos librerías compartidas. No las necesitaremos en esta ocasión.

Ahora compilamos los paquetes uno a uno, siguiendo la recomendación de *Linux from scratch.*

```
# make -C gnulib-lib
# make -C intl pluralx.c
# make -C src msgfmt
# make -C src msgmerge
# make -C src xgettext
```

Ahora vamos a instalar los programas ya compilados "msgfmt, msgmerge y xgettext" en "*/tools/bin*".

```
# cp -v src/{msgfmt,msgmerge,xgettext} /tools/bin
```

Grep-3.1

Contiene programas para hacer búsquedas a través de los ficheros.

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar -xvf grep-3.1.tar.xz
# cd grep-3.1
# ./configure --prefix=/tools
# make
# make check
# make install
```

En función del host es posible que aparezcan errores durante los tests.

Gzip-1.9

Contiene programas para comprimir y descomprimir ficheros.

Descomprimimos y accedemos al fichero.

```
# tar -xvf gzip-1.9.tar.xz
# cd gzip-1.9
```

Como ya hemos hecho en paquetes anteriores vamos a solucionar algunos problemas conocidos de glibc con el siguiente código en el terminal.

```
sed -i 's/IO_ftrylockfile/IO_EOF_SEEN/' lib/*.c
echo "#define _IO_IN_BACKUP 0x100" >> lib/stdio-impl.h
```

Una vez solucionado esto, configuramos el script de configuración, compilaremos, haremos los test e instalaremos.

```
# ./configure --prefix=/tools
# make
# make check
# make install
```

Make-4.2.1

Contiene programas para comprimir y descomprimir ficheros.

Descomprimimos y accedemos al fichero.

```
# tar -xvf make-4.2.1.tar.bz2
# cd make-4.2.1
```

Como ya hemos hecho en paquetes anteriores vamos a solucionar algunos problemas conocidos de *glibc* con el siguiente código en el terminal.

```
sed -i '211,217 d; 219,229 d; 232 d' glob/glob.c
```

Una vez solucionado esto, configuramos el script de configuración

```
# ./configure --prefix=/tools --without-guile
```

Compilamos, hacemos los test e instalamos.

```
# make
# make check
# make install
```

En función del host es posible que los test generen errores.

Patch-2.7.6

Este paquete contiene herramientas de creación o modificación de ficheros aplicando ficheros *patch* creados habitualmente por el programa *diff*.

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar -xvf patch-2.7.6.tar.xz
# cd patch-2.7.6
# ./configure --prefix=/tools
# make
# make check
# make install
```

En función del host es posible que los tests den errores críticos, no es un problema.

Perl-5.28.0

Contiene las librerías básicas de Perl.

Descomprimimos y accedemos al directorio.

```
# tar -xvf perl-5.28.0.tar.xz
# cd perl-5.28.0
```

Configuramos le script de configuración.

```
# sh Configure -des -Dprefix=/tools -Dlibs=-lm -Uloclibpth –Ulocincpth
```

-des
Es una combinación de tres opciones: -d usa la opción por defecto para todos los elementos; -e garantiza la realización de todas las tareas; -s silencia todos los mensajes de salida que no sean esenciales.

-Uloclibpth amd -Ulocincpth
Quita los valores de algunas variables que provocan que la configuración busque componentes localmente instalados que podrías existir en el sistema host.

Ahora pasamos a compilar el paquete.

```
# make
```

Solo necesitamos instalar algunas de las librerías así que lo haremos a mano.

```
# cp -v perl cpan/podlators/scripts/pod2man /tools/bin
# mkdir -pv /tools/lib/perl5/5.28.0
# cp -Rv lib/* /tools/lib/perl5/5.28.0
```

Sed-4.5

Este paquete contiene un *stream editor.*

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar –xvf sed-4.5.tar.xz
# cd sed-4.5
# ./configure --prefix=/tools
# make
# make check
# make install
```

En función del host es posible que los tests den errores críticos, no es un problema.

Tar-1.30

Contiene programas de archivación.

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar –xvf tar-1.30.tar.xz
# cd tar-1.30
# ./configure --prefix=/tools
# make
# make check
# make install
```

En función del host es posible que los tests den errores críticos, no es un problema.

Texinfo-6.5

Contiene programas de lectura, escritura y conversión de páginas de información.

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar –xvf tar -xvf texinfo-6.5.tar.xz
# cd texinfo-6.5
# ./configure --prefix=/tools
# make
# make check
# make install
```

En función del host es posible que los tests den errores críticos, no es un problema.

Util-linux-2.32.1

Este paquete contiene programas útiles varios.

Empezamos por descomprimir el paquete y acceder a la carpeta

```
# tar -xvf util-linux-2.32.1.tar.xz
# cd util-linux-2.32.1
```

Configuramos el script de compilación.

```
./configure --prefix=/tools          \
      --without-python               \
      --disable-makeinstall-chown    \
      --without-systemdsystemunitdir \
      --without-ncurses              \
      PKG_CONFIG=""
```

--without-python
Esta opción deshabilita Python si está instalado en el sistema host.

--disable-makeinstall-chown
Evita que se utilice el comando *chown* durante la instalación. No es necesario cuando instalamos en el directorio *"/tools".*

--without-ncurses
Esta opción deshabilita el uso de las librerías *ncurses* durante el proceso de compilación.

--without-systemdsystemunitdir
En sistemas que usan *systemd*, este paquete trata de instalar un fichero específico *systemd* en un directorio inexistente en "*/tools*". Esta opción deshabilita esta acción.

PKG_CONFIG=""
Establecer esta variable de entorno evita que se añadan características innecesarias que podrían estar disponibles en el sistema host.

Compilamos e instalamos.

```
# make
# make install
```

Xz-5.2.4

Este paquete contiene programas para comprimir y descomprimir ficheros.

Descomprimimos, accedemos al directorio, configuramos el script de compilación, compilamos, hacemos tests e instalamos.

```
# tar –xvf  tar -xvf xz-5.2.4.tar.xz
# cd xz-5.2.4
# ./configure --prefix=/tools
```

```
# make
# make check
# make install
```

En función del host es posible que los tests den errores críticos, no es un problema.

12.3. Instalación del sistema

Llegados a este punto ya tenemos un sistema temporal funcional sobre el que podemos trabajar.

Ahora vamos a instalar los paquetes propios de Debian. Lo primero que vamos a hacer es descargar todos los paquetes. Podemos obtener los enlaces de la propia web de Debian.

Los paquetes a descargar son los siguientes:

- apt
- debian-archive-keyring
- dpkg
- gcc-4.9-base
- gnupg
- gpgv
- libacl1
- libapt-pkg4.12
- libattr1
- libbz2-1.0
- libc6
- libgcc1
- liblzma5
- libpcre3
- libreadline6
- libselinux1
- libstdc++6
- libtinfo5
- libusb-0.1-4
- multiarch-support
- readline-common
- tar
- zlib1g

Lo más recomendable es utilizar el comando es establecer todas las rutas en un documento y utilizar *wget* para descargar todos los paquetes [ANEXO], por ejemplo:

```
# wget --input-file=paquetes --continue --directory-prefix=$LFS/debs
```

12.3.1. Creación de los sistemas de ficheros virtuales

A partir de este punto vamos a trabajar como *root*. Por tanto, vamos a utilizar el usuario con permisos de admistrador para establecer la *password* de *root.* Y accedemos con el usuario root.

```
# sudo passwd
# su
```

Lo primero que haremos será crear los directorios que contienen el *virtual kernel filesystem.* Este sistema de ficheros solamente existe en memoria y se crea dinámicamente cada vez que se carga el kernel.

Cada uno de ellos tiene un propósito diferente. *Devpts* contiene ficheros de dispositivo por cada pseudo terminal del sistema. *Proc* contiene información de cada uno de los procesos en ejecución. *Sysfs* contiene los drivers e información de los dispositivos. *Tmpfs* es espacio libre usable en el que algunos programas podría almacenar información.

Dado que aún no tenemos disponible nuestro kernel vamos a utilizar los del sistema host montando los directorios en las direcciones adecuadas. Cuando nuestro sistema esté completo, el kernel creará estos *filesystems* automáticamente en los lugares apropiados.

Creamos también los dispositivos virtuales "*/dev/console*" y "*/dev/null*" como dispositivos especiales de caracteres con *major* 5 y *minor* 1 para "/dev/*console" y permisos de lectura y escritura para el owner.* M*ajor* 1 y *minor* 3 para "/dev/null" y permiso de lectura y escritura para todo el mundo.

```
# mkdir -pv $LFS/{dev,proc,sys,run}
# mknod -m 600 $LFS/dev/console c 5 1
# mknod -m 666 $LFS/dev/null c 1 3
# mount -v --bind /dev $LFS/dev
# mount -vt devpts devpts $LFS/dev/pts -o gid=5,mode=620
# mount -vt proc proc $LFS/proc
# mount -vt sysfs sysfs $LFS/sys
# mount -vt tmpfs tmpfs $LFS/run
```

Por último, si "*$LFS/dev/shm*" es un *symlink* se crea un directorio a donde apunta.

```
if [ -h $LFS/dev/shm ]; then
      mkdir -pv $LFS/$(readlink $LFS/dev/shm)
fi
```

Ya tenemos todo listo para cambiar a nuestro nuevo sistema utilizando el comando *chroot*.

```
chroot "$LFS" /tools/bin/env -i                                    \
HOME=/root                                                         \
TERM="$TERM"                                                       \
PS1='\[\033[01m\][\[\033[01;34m\]\u@\h\[\033[00m\]\[\033[01m\]]\[\
033[01;32m\]\w\[\033[00m\]\n\[\033[01;34m\]$\[\033[00m\]> '  \
PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin:/tools/sbin    \
/tools/bin/bash --login +h
```

Junto con el comando *chroot* vamos a establecer algunas variables.

HOME = Define el directorio home de usuario root.

TERM = Define el tipo de terminal.

PS1 = Define algunos de los parámetros del *prompt.*

PATH = Define las rutas donde se buscarán los programas.

Por último, lanzaremos el *login* en bash.

12.3.2. Compilación e instalación de dpkg

El programa dpkg nos permitirá instalar todos los paquetes *.deb* que nos hemos descargado. Lo primero que tenemos que hacer es descomprimirlo y acceder al directorio.

```
# tar -xvf dpkg_1.17.27.tar.xz
# cd dpkg-1.17.27/
```

Ahora vamos a configurar el script de compilación para *dpkg*, compilamos e instalamos. Pero dado que configure va a utilizar rutas por defecto, primero crearemos enlaces simbólicos en bash y sh.

```
# mkdir /bin
# ln /tools/bin/bash /bin/bash
# ln /tools/bin/bash /bin/sh
# ./configure                 \
      --prefix=/usr           \
      --sysconfdir=/etc       \
      --localstatedir=/var    \
      --build=x86_64-unknown-linux-gnu
# make
# make install
```

Ya disponemos de dpkg, dado que aún no hemos instalado ningún paquete, creamos una base de datos vacía con el comando *touch.*

```
# touch /var/lib/dpkg/status
```

12.3.3. Descarga e instalación de paquetes .deb

Ya tenemos todo listo para comenzar a instalar los paquetes. Estos siguen un árbol de dependencias por lo que tenemos que instalarlos en el orden específico. En particular *libc6, libgccl* y *multiarch-support*, son codependientes unas de otras por lo que pueden dar problemas en algunas ocasiones. Utilizaremos la opción *–force-depends* para que se instalen pese a los problemas de dependencias y una vez con los 3 paquetes instalados utilizamos *–configure –a* para que lleve a cabo la configuración de los paquetes que se hayan quedado sin configurar.

En ocasiones en necesario crear algunos ficheros o directorios que son requeridos en el proceso de instalación o configuración de los paquetes. Para ello utilizaremos *touch* o *mkdir*.

```
# dpkg -i gcc-4.9-base_4.9.2-10+deb8u2_amd64.deb
# touch /tools/etc/ld.so.conf
# dpkg --force-depends -i libgcc1_4.9.2-10+deb8u2_amd64.deb
# dpkg --force-depends -i multiarch-support_2.19-18+deb8u10_amd64.deb
# dpkg --force-depends -i libc6_2.19-18+deb8u10_amd64.deb
# dpkg --configure -a
# dpkg -i libbz2-1.0_1.0.6-7+b3_amd64.deb
# dpkg -i zlib1g_1.2.8.dfsg-2+b1_amd64.deb
# dpkg -i gpgv_1.4.18-7+deb8u5_amd64.deb
# dpkg -i debian-archive-keyring_2017.5~deb8u1_all.deb
# dpkg -i libattr1_2.4.47-2_amd64.deb
# dpkg -i libacl1_2.2.52-2_amd64.deb
# dpkg -i libpcre3_8.35-3.3+deb8u4_amd64.deb
# dpkg -i libselinux1_2.3-2_amd64.deb
# mkdir /var/log
# touch /var/log/alternatives.log
# dpkg -i tar_1.27.1-2+deb8u1_amd64.deb
# dpkg -i liblzma5_5.1.1alpha+20120614-2+b3_amd64.deb
# dpkg -i dpkg_1.17.27_amd64.deb
# dpkg -i readline-common_6.3-8_all.deb
# dpkg -i libtinfo5_5.9+20140913-1+deb8u3_amd64.deb
# dpkg -i libreadline6_6.3-8+b3_amd64.deb
# dpkg -i libusb-0.1-4_0.1.12-25_amd64.deb
# dpkg -i gnupg_1.4.18-7+deb8u5_amd64.deb
# dpkg -i libstdc++6_4.9.2-10+deb8u2_amd64.deb
# dpkg -i libapt-pkg4.12_1.0.9.8.4_amd64.deb
# dpkg -i apt_1.0.9.8.4_amd64.deb
```

12.3.4. Crear ficheros de configuración del sistema

Vamos a crear un documento *resolv.conf* donde establecer los DNS para la resolución de nombre.

```
cat > /etc/resolv.conf << "EOF"
nameserver 8.8.8.8
nameserver 8.8.4.4
EOF
```

Ahora definimos los repositorios a los que se conectará *apt*, estableceremos los oficiales de Debian 8.

```
 cat > /etc/apt/sources.list << "EOF"
# Debian Jessie main repos
deb http://httpredir.debian.org/debian/ jessie main
deb-src http://httpredir.debian.org/debian/ jessie main

#Debian Jessie security repos
deb http://security.debian.org/ jessie/updates main
deb-src http://security.debian.org/ jessie/updates main

# non-free plugins
deb http://http.debian.net/debian/ jessie non-free contrib main

# jessie-updates, previously known as 'volatile'
deb http://httpredir.debian.org/debian/ jessie-updates main
deb-src http://httpredir.debian.org/debian/ jessie-updates main
EOF
```

Creamos el fichero para la resolución interna de direcciones.

```
cat > /etc/hosts << "EOF"
127.0.0.1       localhost

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
EOF
```

Creamos el fichero que establece el nombre de la máquina.

```
cat > /etc/hostname << "EOF"
arkadium
EOF
```

Creamos el directorio "*/tmp*" para garantizar que no hay errores al crear ficheros temporales.

```
# mkdir /tmp
```

Actualizamos los registros de claves de *debían-archive-keyring.*

```
# apt-key update
```

Por último, actualizaos la información de los repositorios.

```
# apt-get update
```

12.3.5. Instalación y creación de usuarios

Antes de instalar más software, debemos estar seguros de que usuarios, grupos, *passwords* y todos los mecanismos de autenticación funcionan como se espera. Esto es porque algunos paquetes requieren de la creación de usuarios o grupos como parte de su instalación. Si los mecanismos de gestión de usuarios no funcionan correctamente, el proceso de instalación de algunos paquetes podría fallar.

Debianutils

Intalaremos *debianutils* que proporciona el comando *tempfile*. Este comando lo utiliza uno de los scripts de instalación de *base-passwd*. Sin *debianutils* la instalación fallaría.

```
# apt-get install debianutils
```

Base-passwd

Instalaremos *base-passwd* para garantizar al menos los ficheros "*/etc/passwd*" y "*/etc/group*" los cuales son los mismos en todos los sistemas Debian. Los crea ejecutando *update-passwd* durante la instalación.

```
#apt-get install base-passwd
```

Creando "/etc/shadow" y "/etc/gshadow"

Debemos crear manualmente los ficheros "/etc/shadow" y "/etc/gshadow" dado que la configuración del paquete *passwd* fallará si no existen.

```
#touch /etc/shadow /etc/gshadow
```

Login

Este paquete, entre muchas otras cosas, nos da la capacidad de establecer nuevas sesiones en el sistema con *login*, escalado de privilegios con *su*, el módulo de autenticación para ambos mencionados, un falso shell "*/bin/nologin*" y "*/etc/login.defs*", fichero esencial para la creación de grupos.

```
#apt-get install login
```

Passwd

Ofrece las herramientas y los ficheros de configuración necesarios para crear y manipular la información de grupos y usuarios.

```
# apt-get install passwd
```

Adduser

Instalaremos también el paquete *adduser* dado que nos proporciona el fichero "*/etc/adduser.conf*" por defecto, el cual es necesario para crear adecuadamente a los usuarios.

```
# apt-get install adduser
```

Estableciendo la password y las entradas del fichero shadow

Llegados a este punto, nuestro sistema es perfectamente capaz de manipular usuarios y grupos. Por tanto solo nos queda establecer una *password* al usuario root.

```
# passwd root
```

Y convertir las entradas de nuestro "*/etc/passwd*" en las entradas de "*/etc/shadow*".

```
# pwconv
```

12.3.6. Corrigiendo el terminal y añadiendo utilidades de lectura y edición

Nuestro terminal aún no dispone de todas las funcionalidades que uno espera de un terminal. Vamos a instalar las librerías adecuadas y crear las configuraciones necesarias para que nuestra terminal sea capaz de llevar a cabo lo mínimo esperable de una terminal estándar.

Creando el fichero "/etc/inputrc"

El fichero "*/etc/inputrc*" es un fichero de configuración global usado por la librería *libreadline6* la cual es usada por la mayoría de *shells* para poder manejar algunas de las situaciones especiales del teclado, como por ejemplo, cómo comportarse por defecto al pulsar la tecla *home* o *end*. Sin este fichero algunas de las combinaciones de teclas más comunes no funcionan.

Dado que este fichero no se crea por defecto al instalar la librería *libreadline6*, lo vamos a crear nosotros.

```
cat > /etc/inputrc << "EOF"
# /etc/inputrc - global inputrc for libreadline
# See readline(3readline) and `info rluserman' for more
information.

# Be 8 bit clean.
set input-meta on
set output-meta on

# To allow the use of 8bit-characters like the german umlauts,
uncomment
# the line below. However this makes the meta key not work as a
meta key,
# which is annoying to those which don't need to type in 8-bit
characters.

# set convert-meta off

# try to enable the application keypad when it is called.  Some
systems
# need this to enable the arrow keys.
# set enable-keypad on

# see /usr/share/doc/bash/inputrc.arrows for other codes of arrow
keys

# do not bell on tab-completion
# set bell-style none
# set bell-style visible

# some defaults / modifications for the emacs mode
$if mode=emacs


# allow the use of the Home/End keys
"\e[1~": beginning-of-line
"\e[4~": end-of-line


# allow the use of the Delete/Insert keys
"\e[3~": delete-char
"\e[2~": quoted-insert
```

```
# mappings for "page up" and "page down" to step to the
beginning/end
# of the history
# "\e[5~": beginning-of-history
# "\e[6~": end-of-history

# alternate mappings for "page up" and "page down" to search the
history
# "\e[5~": history-search-backward
# "\e[6~": history-search-forward

# mappings for Ctrl-left-arrow and Ctrl-right-arrow for word moving
"\e[1;5C": forward-word
"\e[1;5D": backward-word
"\e[5C": forward-word
"\e[5D": backward-word
"\e\e[C": forward-word
"\e\e[D": backward-word

$if term=rxvt
"\e[7~": beginning-of-line
"\e[8~": end-of-line
"\eOc": forward-word
"\eOd": backward-word
$endif

# for non RH/Debian xterm, can't hurt for RH/Debian xterm
# "\eOH": beginning-of-line
# "\eOF": end-of-line

# for freebsd console
# "\e[H": beginning-of-line
# "\e[F": end-of-line

$endif
EOF
```

Librerías y binarios de ncurses

Un gran número de líneas de comando requieren de capacidades relacionadas con las librerías *ncurses* para interacción con el usuario basada en texto texto a través del terminal. Incluye programas como *less* o *nano.* Sin estas librerías, no se mostrarían algunos elementos correctamente. Para evitar este tipo de problemas instalaremos las librerías *ncurses.*

```
# apt-get install ncurses-base ncurses-bin ncurses-doc
```

Dialog

Es un módulo *Perl* que algunos scripts utilizan para ofrecer una interfaz basada en texto utilizada durante la instalación o la configuración. Puede que algunos paquetes lancen *warnings* por no estar instalado en el sistema, así que vamos a instalarlo.

```
# apt-get install dialog
```

Less, vim y nano

Ahora que hemos instalado la mayoría de las librerías y utilidades necesarias para el terminal, vamos a instalar algunos de los editores más básicos y conocidos.

```
# apt-get install less vim nano
```

12.3.7. Creando el sistema de ficheros jerárquico de un sistema Debian

Vamos a crear la estructura de directorios estándar de un sistema Debian. Esto se puede realizar fácilmente con la instalación de *base-files.* Antes que nada, debemos eliminar el directorio "*/var/mail*" para que no de ningún tipo de problema o no se complete correctamente.

```
# rm -rf /var/mail
# apt-get install base-files
```

12.3.8. Instalando la documentación del sistema

Cualquier sistema Linux dispone generalmente de una base de datos completa de manuales accesible con el comando *man.* La mayoría de los programas que hemos instalado ya incluyen su propia documentación en los directorios adecuados. Instalaremos *man* para poder disponer de toda la documentación de nuestro sistema.

```
# apt-get install man
```

12.3.9. Instalando el resto de paquetes esenciales

Ya hemos instalado la mayoría de los paquetes. Solo nos quedan por instalar lo paquetes marcado como *essential* por el equipo de desarrollo de Debian. Algunos de ellos son completamente esenciales para la gestión del sistema.

```
# apt-get install bash bsdutils coreutils dash diffutils
e2fsprogs findutils grep gzip hostname libc-bin init mount perl-
base sed sysvinit-utils tar util-linux
```

Durante la instalación de estos paquetes aparecerán cuadros de diálogo preguntando por nuestra zona horaria para completar las configuraciones.

bash:	*Shell* estándar de *Linux*.
bsdutils:	Incluye algunos binarios, alguno de los más importantes son *renice*, necesario para cambiar la prioridad de los procesos y *logger* que se usa para interactuar con el módulo de sistema *syslog.*
coreutils:	El grupo más esencial de programas necesarios para hacer útil un s*hell*.
dash:	Una versión más rápida de sh, principalmente utilizada por algunos scripts-
diffutils:	Incluye utilidades para comparar el contenido de los ficheros.
e2fsprogs:	Incluye herramientas para trabajar con la familia de sistemas de ficheros *ext.*
findutils:	Incluye herramientas para encontrar ficheros.
grep:	La herramienta grep permite encontrar cadenas de caracteres dentro de ficheros o pipes.
gzip:	Proporciona la herramienta *gzip*, usada para codificar ficheros.
hostname:	Proporciona un conjunto de herramientas para manipular el nombre del equipo.
libc-bin:	Proporciona una implementación GNU de las librerías estándar de C. Esencial para la creación y uso de programas.
init:	Ofrece el conjunto de programas de inicialización del sistema para Debian.
mount:	Proporciona las herramientas de sistema estándar para montaje y desmontaje de sistemas de ficheros.
perl-base:	Proporciona el lenguaje de programación *perl.*
sed:	Proporciona el lenguaje de programación *sed* normalmente utilizado para editar texto.
sysvinit-utils:	Proporciona *system-v* y otras utilidades.
tar:	Proporciona el programa *tar* utilizado para almacenar y comprimir ficheros.
util-linux:	Incluye algunas de las herramientas elementales del sistema.

12.3.10. Instalando el kernel

Tenemos 2 opciones. Compilar un kernel o instalar uno desde los repositorios de Debian. En nuestro caso tomaremos la segunda opción.

Utilizaremos el comando *apt-cache* para encontrar las iso en los repositorios de debían y seleccionaremos una para descargarla e instalarla con *apt-get.*

```
# apt-cache search linux-image
```

Seleccionaremos linux-image-4.9-amd64 en nuestro caso, por tanto:

```
# apt-get install linux-image-4.9-amd64
```

Además queremos poder cargar y descargar módulos en el kernel por lo que necesitaremos instalar *kmod.*

```
# apt-get install kmod
```

12.3.11. Instalación y configuración de grub

Lo más probable es que ya dispongamos de un grub instalado que se ha instalado con *Ubuntu*. Solamente necesitamos configurarlo para que detecte el nuevo sistema. Para ello utilizarmos el comando:

```
# grub-mkconfig
```

12.3.12. Instalación y configuración de librerías gráficas (awesome) y audio

Llegados a este punto, nuestro sistema operativo Debian ya es completamente funcional y puede arrancar por sí mismo.

Arrancamos nuestro sistema y accedemos como root para evitar problemas de ejecución y permisos. Establecemos una IP haciendo una solicitud al servidor DHCP al que estemos conectados con el comando:

```
# dhclient eth0
```

Pasaremos a instalar un gestor de ventanas llamado *awesome* que además de facilitarnos algunas de las tareas de ahora en adelante, muchas de sus dependencias son programas y librerías gráficas que necesitaremos más adelante. También necesitaremos instalar el paquete de programas de *Xorg*, que son también librerías gráficas.

Para realizar la instalación utilizaremos los comandos:

```
# apt-get install awesome
# apt-get install xorg
# apt-get install alsa
```

Veremos que el número de dependencias en muy grande, pero es correcto.
Una vez instalados ambos paquetes, podemos lanzar *awesome* con el comando:

```
# startx awesome
```

A partir de este momento tenemos una interfaz gráfica con la que poder trabajar en diferentes ventanas.

12.4. Implementación de programas específicos

Una vez tenemos una versión personalizada de Debian 8 en funcionamiento, las librerías gráficas adecuadas y *awesome* como gestor de ventanas vamos a trabajar en lo que realmente hace de este sistema operativo, realmente *Arkadium*.

Básicamente hay 3 elementos que forman en núcleo de la experiencia que debe brindar *Arkadium* como sistema operativo.

El primero de estos elementos es *M.A.M.E*. Un emulador que permitirá ejecutar los juegos como si de una máquina arcade se tratase.

El segundo elemento es *EmulationStation*. Un frontend que ejercerá de intermediario entre *M.A.M.E* y el usuario.
La versión oficial de *EmulationStation* lleva más de 5 años sin actualizarse mientras que hay 2 forks muy conocidos y continuamente actualizados por la comunidad.

recalboxEmulationStation y *retropieEmulationStation*. Nos quedaremos con esta segunda versión que dispone de unos scripts de apagado predefinidos y un modo quiosco muy interesante que impide que el usuario pueda salir del entorno del programa.

Y el tercer elemento es la configuración del sistema, de forma que todos los procesos sean lo más transparente posible para el usuario.

12.4.1. Compilación e instalación de M.A.M.E

M.A.M.E es un software opensource y está disponible gratuitamente en los repositorios públicos de *github*. Para poder descargar el código, necesitaremos, antes que nada, instalar *git*. Para ello utilizaremos el comando:

```
# apt-get install git
```

Lo primero que haremos será instalar las dependencias, pero una de ellas no podremos cumplirla porque no está en los repositorios de *Debian* 8.

Para poder descargar gcc-5 necesitaremos añadir antes el repositorio de testing/sid de *debian*. Para añadir el repositorio, ejecutamos el comando:

```
# vi /etc/apt/sources.list
```

Esto nos abrirá un editor, añadimos la línea "deb http://ftp.de.debian.org/debian sid main", guardamos y actualizamos repositorios con el comando:

```
# apt-get update
```

Ahora ahora ya podemos instalar gcc-5 de la forma común:

```
# apt-get install gcc-5
```

A continuación, instalamos el resto de las dependencias.

```
# apt-get install build-essential libsdl2-dev libsdl2-ttf-dev
libfontconfig-dev qt5-default
```

Ya lo tenemos listo. Ahora vamos a establecer una estructura de carpetas para albergar los ficheros fuente y compilar.

estructura de carpetas

/arkadium
/mame
/emustation
/games
/bios
/neo

```
# cd /
# mkdir arkadium arkadium/mame arkadium/emustation
arkadium/games arkadium/games/bios arkadium/games/neo
```

Nos posicionamos en la carpeta "*/arkadium/mame*" y clonamos el repositorio de *M.A.M.E*.

```
# git clone https://github.com/mamedev/mame.git .
```

Ahora ya tenemos todo lo necesario para iniciar la compilación.
M.A.M.E como software tiene 2 modos. El modo arcade y el modo mes, que viene a ser una evolución del modo arcade con muchísimas nuevas opciones y elementos. En nuestro caso, dado que buscamos una versión lo más ligera posible, compilaremos el modo arcade.

```
# make SUBTARGET=arcade
```

Este proceso puede llevar bastante tiempo.

12.4.2. Configuración de M.A.M.E

Ya tenemos *M.A.M.E* compilado y funcionando. Podemos probarlo siempre que tengamos *awesome* activo, usando el comando:

```
# /arkadium/mame/mamearcade64
```

Ahora bien, debemos configurar el programa para que funcione de la forma en que nosotros esperamos.
Lo primero que haremos será generar un fichero de configuración genérico utilizando el comando:

```
# /arkadium/mame/mamearcade64 -cc
```

Nos aparecerá un fichero llamado mame.ini en el directorio que es el fichero de configuración por defecto generado.
Dado que es muy probable que ejecutemos el software sobre máquinas antiguas, activaremos el frameskip automático y desactivaremos la información del juego para que se ejecute de forma automática.

La información del juego es un cuadro de información que bloquea la ejecución del juego esperando que el usuario interactúe con la máquina. No nos interesa este tipo de comportamiento.

También debemos establecer las rutas de las roms, y las bios.

Para modificar el fichero de configuración utilizaremos el comando:

```
# vi /arkadium/mame/mame.ini
```

Esto no abrirá el editor y modificaremos únicamente los puntos indicados de forma que, el resto de las opciones quedarán por defecto.

```
#
# CORE SEARCH PATH OPTIONS
#
rompath    /arkadium/games/roms/bios;/arkadium/games/neo;/arkadium/games/roms

#
# CORE PERFORMANCE OPTIONS
#
autoframeskip    1

#
# CORE MISC OPTIONS
#
skip_gameinfo    1
```

12.4.3. Compilación e instalación de emulation station (RetroPie edition)

De nuevo, lo primero que haremos será instalar las dependencias indicadas por los desarrolladores en la página de github, con el comando:

```
# apt-get install libsdl2-dev libfreeimage-dev libfreetype6-dev
libcurl4-openssl-dev libasound2-dev libgl1-mesa-dev build-
essential cmake fonts-droid-fallback libvlc-dev libvlccore-dev
vlc-bin libcec-dev rapidjson-dev
```

Ahora clonamos el repositorio con la opción "--recursive" dado que hay subrepositorios.

```
# cd /arkadium/emustation
# git clone --recursive
https://github.com/RetroPie/EmulationStation.git
```

Ahora ya con todo listo podemos compilar *emulationStation*, primero debemos generar el *makefile* con el programa *cmake*.

```
# cmake .
# make
```

12.4.4. Configurar emulation station

El fichero de configuración estándar de *emulationStation* se genera de forma automática durante la primera ejecución. El programa nos muestra un mensaje indicando que no hay una configuración válida y cierra el programa.

De forma que, con *awesome* iniciado, ejecutamos *emulationStation* para generar el fichero de configuración:

```
# /arkadium/emustation/emulationstation.sh
```

El fichero de configuración se genera en el directorio *"~HOME/.emulationstation/es_system.cfg".* En nuestro caso, dado que hemos utilizado constante el usuario root para evitar cualquier tipo de problema por lo que encontraremos el fichero de configuración en "*/root/.emulationstation*".

Debemos modificarlo para los propositos de nuestro sistema de forma que resultará como en el cuadro de texto:

/root/.emulationstation/es_system.cfg

```
<systemList>
      <system>
            <name>arcade</name>
            <fullname>Arkadium system</fullname>
            <path>/arkadium/games/roms/neo</path>
            <extension>.zip</extension>
            <command>/arkadium/mame/mamearcade64 %ROM%</command>
            <platform>Arcade</platform>
            <theme>arcade</theme>
      </system>
</systemList>
```

El fichero de configuración de *emulationstation* se basa en lenguaje de etiquetas y es relativamente simple. Podemos encontrar información sobre ello en la web oficial, en "Getting Started".

12.4.5. Creación tema personalizado

EmulationStation utiliza un sistema de temas para personalizar la experiencia. En nuestro caso, crearemos un nuevo tema específico para *Arkadium* que ayudará a darle un toque único al sistema.

Los temas se desarrollan utilizando *XML* con etiquetas preestablecidas. Podemos encontrar algunos tutoriales bien desarrollados en la propia web de *emulationstation* y docenas de temas ya creados en los foros.

Para crear nuestro tema utilizaremos el tema por defecto como base y lo modificaremos hasta conseguir el resultado esperado. Además, utilizaremos alguna herramienta para crear las imágenes para personalizar el tema.

Los documentos e imágenes que forman el tema creado para *Arkadium* están disponibles en el **Anexo** y adjuntas al trabajo.

12.4.6. Configurar controles

La primera vez que iniciemos *emulationstation* nos aparecerá un asistente de configuración de los controles para el *frontend*. Esta configuración no se aplica al control de los juegos. Solamente se aplica al control de los menús.

12.5. Personalización de la distribución

Llegados a este punto, el sistema debería ser completamente funcional, arrancando manualmente los programas. Ahora vamos a configurar el sistema para automatizar todo el proceso de arranque de las aplicaciones y establecer un aspecto visual más adecuado.

12.5.1. Autologin

Una de las cosas más importantes es que el *login* del root de haga de forma automática ya que la máquina no tendrá un teclado conectado para poder hacer el *login*. Además, esta característica dota de una mayor transparencia al sistema.

Para automatizar el *login* crearemos un nuevo fichero con:

```
# systemctl edit getty@.service
```

Dentro del nuevo fichero escribiremos las siguientes líneas y guardaremos con el nombre por defecto. El nuevo fichero será /etc/systemd/system/getty@.service.d/override.conf.

```
[Service]
ExecStart=
ExecStart=-/sbin/agetty --noclear --autologin root %I $TERM
```

12.5.2. Personalización de awesome

Awesome establece su configuración en base al fichero "*/etc/xdg/awesome/rc.lua*".

En *Arkadium* queremos que *awesome* sea los más transparente posible para el usuario. Trataremos de ocultar todo lo posible los elementos innecesarios.

Lo primero que haremos será modificar el fichero de configuración comentando o eliminando las siguientes líneas, que son las que establecen la barra superior.

```
    -- Create the wibox
    s.mywibox = awful.wibar({ position = "top", screen = s })

    -- Add widgets to the wibox
    s.mywibox:setup {
        layout = wibox.layout.align.horizontal,
        { -- Left widgets
            layout = wibox.layout.fixed.horizontal,
            mylauncher,
            s.mytaglist,
            s.mypromptbox,
        },
        s.mytasklist, -- Middle widget
        { -- Right widgets
            layout = wibox.layout.fixed.horizontal,
            mykeyboardlayout,
            wibox.widget.systray(),
            mytextclock,
            s.mylayoutbox,
        },
    }
```

Además de modificar el script cambiaremos el fondo de pantalla por defecto de *awesome*. El fondo de pantalla está definido en el tema por defecto que lo coge de la ruta "/usr/share/awesome/themes/default/background.png". Lo substituiremos por el fichero background.png adjunto, también lo podemos encontrar en el anexo.

Por último añadiremos una última línea al fichero de configuración de *awesome* para que se inicie automáticamente *emulationstation*.

```
Awful.util.spawn_with_shell("/arkadium/emustation/emulationstation.
sh --no-splash –force-kiosk")
```

12.5.3. Scripts de inicio

Estableceremos un script que se ejecute al realizar el *login* en el sistema. Este script dejará el sistema listo para la ejecución de los programas. A continuación, mostramos el script y explicamos cada uno de los comandos.

/etc/profile.d/autostart.sh

```
#!/bin/bash

#Establecemos la configuración de los módulos USB
rmmod usbhid
modprobe usbhid quirks=0xYYYY:0xZZZZ:0x00000040
alsactl init
amixer set Master 100
startx awesome
```

Drivers

Es posible que Linux ya disponga del driver para el correcto funcionamiento del dispositivo de control. Pero lo más probable es que, aunque funcione no lo haga como debería.

Para asegurarnos de que funciona correctamente debemos cargar el módulo de la controladora USB con los *quirks* adecuados. Los *quirks* son códigos de identificación del dispositivo.

Para conocer los *quirks* de nuestros dispositivos usb, utilizaremos el comando:

lsusb

Obtendremos una lista de dispositivos conectados al equipo con el formato siguiente:

```
Bus 001 Device 001: ID YYYY:ZZZZ Nombre del dispositivo
```

Los código YYYY:ZZZZ son los código de producto y fabricante y serán los quirks que utilizaremos al cargar el módulo en el kernel de la siguiente forma:

```
# rmmod usbhid
# modprobe usbhid quirks=0xYYYY:0xZZZZ:0x00000040
```

Configuración del sonido

Iniciamos y establecemos el volumen el volumen para garantizar que se escucha correctamente. En nuestro caso lo estableceremos en 100% dado el hardware en el que hemos realizado el proyecto pero debe adaptarse a las necesidades del hardware utilizado.

```
# alsactl init
# amixer set Master 100
```

Awesome (autoarranque de awesome)

Por último, iniciaremos el entorno gráfico para poder ejecutar el resto de aplicaciones que requieren del uso de librerías gráficas.

```
# startx awesome
```

12.5.4. Splash screen arkadium

La personalización de la splash screen se realiza utilizando un programa llamado Plymouth, dicho programa se instala con "*apt-get install playmouth*" y en primera instancia no debería requerir nada más allá de la propia configuración del tema para *Arkadium*.

Sin embargo, hay varios problemas documentados con Debian 8 que incluyen pantalla apagada, problemas en el input del terminal, problemas con la resolución y problemas con algunos entornos gráficos.

Debido a que durante algunas pruebas se han producido problemas intermitentes especialmente de pantallas en negro, o sistema que no termina de estar operativo sin una causa aparente, he tomado la decisión de prescindir de esta característica que es puramente estética, al menos en esta primera versión de *Arkadium.*

12.5.5. Configuración grub autoseleccion

Por último, para conseguir mayor transparencia de cara al usuario, configuraremos grub para que seleccione el sistema *Arkadium* de forma automática. Para ello modificaremos el fichero de configuración de *grub*. Dicho fichero de configuración lo encontraremos en "/etc/default/grub" del sistema donde lo instalamos. En nuestro caso, el sistema *Ubuntu* que instalamos al principio de todo.

Para modificar este fichero tenemos 2 opciones. O bien accedemos al sistema *Ubuntu* a través de *grub*, reiniciando el equipo, o bien montamos la partición en nuestro sistema.

En el segundo caso utilizaremos el comando *mount* para montar la partición

```
# mkdir Ubuntu
# sudo mount -v -t ext4 /dev/sda2 ./ubuntu
# vi ./ubuntu/etc/default/grub
```

Dentro debemos modificar 2 lineas.

```
GRUB_DEFAULT=1
GRUB_TIMEOUT=0
```

GRUB_DEFAULT indica la posición de la lista de *grub* que aparecerá seleccionada por defecto. GRUB_TIMEOUT indica el tiempo que esperará al input del usuario. Poniendo el tiempo a 0, grub no aparecerá y lanzará la opción seleccionada por defecto.

13. Conclusión

Cuando llevamos a cabos proyectos personales como por ejemplo máquinas arcade caseras, en el momento de seleccionar el software no siempre encontramos un software que

cumple con todas nuestras necesidades. En ocasiones, sí encontramos softwares que cumplan con nuestras necesidades pero que además tienen docenas y docenas de funciones que no resultan necesarias para nosotros. Entones, ¿cómo conseguimos el software adecuado para nuestro proyecto?

En este proyecto, *Arkadium*, vemos como utilizando software libre ya existente podemos desarrollar un sistema que cumpla las necesidades particulares de nuestro proyecto. De esta forma recuperamos, en este caso, algunos de los videojuegos clásicos de la historia del entretenimiento y no solamente eso sino, la forma de jugarlos, que es lo que hace única la experiencia.

Esta es solo una forma de hacerlo, existen muchos softwares diferentes que pueden cumplir cometidos similares a los que hacen los programas elegidos para este proyecto. Utilizando otras herramientas podemos conseguir cosas muy similares. Incluso modificando sistemas ya enfocados en la emulación de software de entretenimiento y eliminando algunas de las particularidades que no son necesarios para el proyecto particular en cuestión.

Y no solamente la experiencia de uso que se persigue, sino que desarrollamos un entorno de ejecución que nos permite recrear varias experiencias diferentes gracias a la forma en que interactúan los softwares seleccionados para la realización del proyecto.

En definitiva, con los conocimientos necesarios podemos desarrollar un entorno de ejecución óptimo que cumpla exactamente las necesidades de nuestro proyecto.

14. Bibliografía y referencias

[1] «Historia de los videojuegos», 09-oct-2018. [En línea]. Disponible en: https://www.fib.upc.edu/retro-informatica/historia/videojocs.html. [Accedido: 09-oct-2018].

[2] «Galaxy-Game machine», 09-oct-2018. [En línea]. Disponible en: http://infolab.stanford.edu/pub/voy/museum/pictures/display/5-GG-machine.htm. [Accedido: 09-oct-2018].

[3] «Computer Space - Videogame by Nutting Associates», 09-oct-2018. [En línea]. Disponible en: https://www.arcade-museum.com/game_detail.php?game_id=7381. [Accedido: 09-oct-2018].

[4] H. Lowood, «Videogames in Computer Space: The Complex History of Pong», *IEEE Annals of the History of Computing*, vol. 31, n.º 3, pp. 5-19, jul. 2009.

[5] «Space Invaders», *Vandal*, 09-oct-2018. [En línea]. Disponible en: https://vandal.elespanol.com/retro/space-invaders. [Accedido: 09-oct-2018].

[6] Jimmy, «Máquinas Arcade: Historia y Evolución», *PixFans*, 09-mar-2008. .

[7] D. C. Domínguez, «Las Redes Sociales. Tipología, uso y consumo de las redes 2.0 en la sociedad digital actual», *Documentación de las Ciencias de la Información*, vol. 33, n.º 0, pp. 45-68-68, jul. 2010.

[8] M. G. Ninova, «Comunidades, software social e individualismo conectado», *Athenea digital*, vol. 0, n.º 13, pp. 299-305-305, 2008.

[9] J. S. Clemente Ricolfe, J. M. Buitrago Vera, y E. S. Emper, «Estudio de los factores de compra de productos retro y segmentación del mercado potencial retro», *Contaduría y Administración*, vol. 58, n.º 1, pp. 225-250, ene. 2013.

[10] M. Wolf y S. McQuitty, «Understanding the do-it-yourself consumer: DIY motivations and outcomes», *AMS Review*, vol. 1, n.º 3-4, pp. 154-170, dic. 2011.

[11] «BOE.es - Documento BOE-A-2013-11199», 15-oct-2018. [En línea]. Disponible en: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2013-11199. [Accedido: 15-oct-2018].

[12] «Olvídate de China: Tailandia ya es el nuevo vertedero electrónico del mundo», *El Confidencial*, 15-jun-2018. [En línea]. Disponible en: https://www.elconfidencial.com/mundo/2018-06-15/tailandia-vertedero-basura-electronica-mundo_1578912/. [Accedido: 15-oct-2018].

[13] «Tecnlogia per a tothom | Facultat d'Informàtica de Barcelona», 15-oct-2018. [En línea]. Disponible en: https://txt.upc.edu/. [Accedido: 15-oct-2018].

[14] E. | www.ecoembes.com, «¿Dónde acaban los residuos electrónicos?», *Ecoembes, Revista Circle*, 13-abr-2018. [En línea]. Disponible en: https://www.revistacircle.com/2018/04/13/residuos-electronicos/. [Accedido: 15-oct-2018].

[15] «linuxfromscratch.org – Linux from scratch 7.9», 15-oct-2018. [En línea]. Disponible en: http://www.linuxfromscratch.org/lfs/downloads/7.9/LFS-BOOK-7.9.pdf [Accedido: 15-oct-2018].